681.3
Д-46

Р. Джордейн

СПРАВОЧНИК программиста персональных компьютеров типа IBM PC, XT и AT

681.3
Д-40

Р. Джордейн

СПРАВОЧНИК программиста персональных компьютеров типа IBM PC, XT и AT

Р. Джордейн СПРАВОЧНИК программиста персональных компьютеров типа IBM PC, XT и AT

Справочник программиста персональных компьютеров типа IBM PC, XT и AT

Р. Джордейн

СПРАВОЧНИК программиста персональных компьютеров типа IBM PC, XT и AT

Перевод с английского
Н.В. Гайского

МОСКВА
"ФИНАНСЫ И СТАТИСТИКА"
1992

Programmer's Problem Solver for the IBM PC, XT & AT

ROBERT JOURDAIN

BRADY
New York

ББК 24.4.1
Дж40

Джордейн Р.
Дж40 Справочник программиста персональных компьютеров типа IBM PC, XT и AT: Пер. с англ./Предисл. Н. В. Гайского. -М.: Финансы и статистика, 1992. - 544 с.: ил.

ISBN 5-279-00611-4.

Универсальный справочник пользователя и программиста IBM PC - совместимого персонального компьютера сводит воедино информацию, рассредоточенную в документации, руководствах, учебных пособиях.

Свойства полноты и самодостаточности делают его ценным источником для широкого круга профессиональных пользователей персональных компьютеров.

Д2404010000 -006
010(01) - 92 141 - 91

ББК 24.4.1

ISBN 0-89303-787-7 (США)

ISBN 5-279-00611-4 (СССР)

© 1986 by Brady Books, a division of Simon & Schuster Inc.
© Н.В.Гайский, перевод и предисловие, 1991

Предисловие к русскому изданию

В оригинальном названии предлагаемой советскому читателю книги Роберта Джордейна "Programmer's problem solver" ключевым является слово "solver", переводимое как сборник решений. И действительно, в ней собраны решения наиболее типичных задач, с которыми сталкивается программист при написании программ. Но, как пишет во введении автор, основная цель книги - собрать воедино и систематизировать огромное количество информации по программированию на персональном компьютере типа IBM PC. Ценность книги особенно велика для советского программиста, поскольку далеко не каждый может легко получить доступ к документации фирм-изготовителей вспомогательных микросхем. Но даже при наличии документации вы не всегда можете понять, как решить стоящую перед вами проблему средствами языка высокого уровня, а ведь подавляющее большинство программ пишется на этих языках. Указанная книга поможет вам разобраться и в такой проблеме. Хотя в качестве языка высокого уровня выбран Бейсик, который не слишком распространен в Советском Союзе, тем не менее вы усвоите главное: можно ли реализовать нужную вам функцию в рамках языка высокого уровня или необходимо включать в свою программу ассемблерные подпрограммы.

Хотелось бы отметить и удачный способ расположения материала: каждая глава посвящена устройству или группе близких устройств и описаны все типичные проблемы, возникающие при работе с ними. Несомненно, такой порядок намного более удобен, чем упорядочение по номеру функции операционной системы или алфавиту. Но надо отметить, что в ряде случаев описаны устройства, с которыми советскому программисту вряд ли придется столкнуться в своей работе, например световым пером. С другой стороны, в книге нет информации об устройствах, появившихся за последние несколько лет, в частности, получившем широкое распространение VGA-дисплее. Это связано с тем, что американское издание было выпущено до появления дисплеев такого типа.

Однако мелкие недостатки книги полностью компенсируются ее достоинствами. Во-первых, вы получаете универсальный справочник, охватывающий практически все компоненты и устройства персонального компьютера. Во-вторых, многочисленные

примеры показывают вам путь, по которому программист должен двигаться при решении конкретной задачи. И хотя ни один из примеров не является самостоятельно работающей программой, ценность их несомненна. Как правило, в них отмечены все пустяки и мелочи, уже известные опытному программисту, но которые могут привести к огромным затратам времени и сил, если ранее вы не сталкивались с аналогичными проблемами.

Суммируя вышесказанное, можно выделить две основные группы читателей, для которых книга Джордейна будет особенно полезна. Это те, кто постоянно программирует на персональном компьютере и воспользуется книгой в качестве справочника. И те, кто захотел понять, как работают программы, и для этого выбрал лучший способ - попробовал программировать сам. В заключение хотелось бы дать совет всем, кто начнет изучение программирования с этой книги: не бойтесь ошибиться, экспериментируйте, и вы будете вознаграждены за свои старания пониманием того, как работает ваш персональный компьютер.

Н.В.Гайский

Введение

В наше время программисты являются одной из наиболее новаторских групп. К сожалению их наиболее удачные новшества включают и несколько новых способов потери времени. Бесконечны ужасные истории о программах, для отладки которых требуется в двадцать раз больше времени, чем для написания. И вы можете снова и снова слышать о программах, к которым приходится обращаться вновь и вновь, так как они не были достаточно хорошо продуманы с самого начала. Намного меньше сказано о том, что может оказаться самым емким способом пустой траты времени для изучающих программирование: поиск информации о машине. Многочасовые усилия по установлению простого факта являются настоящим обрядом посвящения для начинающих программистов, которые должны рыться в руководствах до потемнения в глазах.

Типичное следующее утро после этого: глаза, слезящиеся от терминала, метровая стопа смятых распечаток и пара дюжин руководств, рассыпанных по всему полу. Среди них руководства по оборудованию, по операционной системе MS-DOS, по языку программирования, а также описания отдельных микросхем, печатающего устройства и клавиатуры плюс дюжина дополнительных книг, каждая из которых содержит бесценный кусочек информации, понадобившийся в три часа ночи для особо тонкого места в программе.

Поскольку немногие обладают фотографической памятью (а работа с компьютерами может лишить вас остатков памяти), все эти книги действительно необходимы, так как одни и те же вещи приходится искать снова и снова. На первых порах вы можете затрачивать часы и не находить требуемой информации. Даже если вы обнаружили нужное место, то может понадобиться достаточно много времени, чтобы вытянуть искомое из пространного описания для начинающих, или, если, к вашему несчастью, руководство написано на языке суахили, то не меньше чем полдня уйдет на перевод. Хотелось бы иметь одну большую книгу, в которой собрано практически все необходимое, не разбавленное информацией, ненужной для программистов, написанную на среднем уровне сложности, описывающую все компоненты IBM PC и организованную таким способом, чтобы в ней легко было отыскать необходимую информацию. Но слышали ли вы когда-нибудь о такой книге?

Я собрал вместе все эти руководства и описания для тех, кто хочет научиться писать нетривиальные программы, но не может позволить себе тратить массу времени (или 600 - 800 дол., чтобы купить все эти книги). Материал представлен двумя способами. Во-первых, главы делятся по типу описываемого оборудования, их разделы относятся к определенному свойству данного оборудования и включают короткие параграфы, относящиеся к конкретной программистской задаче. Например, один из разделов главы, посвященной выводу на дисплей, относится к курсору и содержит параграфы, описывающие, как позиционировать курсор, менять его форму, включать и выключать его и т.д.

Во-вторых, каждый параграф делится на четыре части (иногда меньше). Сначала идут несколько абзацев, описывающих основные понятия. Затем рассматриваемая задача обсуждается с точки зрения программирования на языках высокого уровня, среднего уровня (прерываний BIOS и DOS) и низкого уровня вспомогательных микросхем, поддерживающих микропроцессор. Кроме того, каждый из разделов начинается с пары страниц, содержащих сведения, необходимые для понимания данного раздела. Эти сведения носят характер обзора и вы можете использовать их, чтобы наметить свой путь изучения книги при первом ее просмотре.

Обсуждение программирования на языке высокого уровня поможет читателю решать конкретные проблемы. Хотя рассматриваемые концепции в равной степени применимы и к Паскалю, и к Си, все примеры приведены на Бейсике. Бейсик выбран отчасти потому, что он является базовым языком для микроЭВМ, отчасти потому, что каждому владельцу IBM PC он доступен, и отчасти потому, что Бейсик фирмы Microsoft по сравнению с другими языками программирования предоставляет наиболее полные средства для использования возможностей оборудования IBM PC. Даже начинающие программировать на Бейсике найдут в книге много полезного. Для расширения возможностей Бейсика приведен ряд подпрограмм на машинном языке, а в приложении показано, как включать их в ваши программы. С помощью этих подпрограмм вы можете делать такие тонкие вещи, как перепрограммирование клавиатуры или создание дополнительных дисплейных страниц для монохромного адаптера.

Решение проблем с помощью прерываний, включенных в операционную систему, будем называть программированием на среднем уровне. Прерывания - это мощные компактные процедуры, выполняющие нудную работу любого компьютера, такую, как перемещение курсора или чтение каталога диска. Это область программистов на языке ассемблера, и все примеры программирования на среднем уровне приведены на языке ассемблера. Но теперь все больше и больше трансляторов с языков высокого уровня предоставляют доступ к прерываниям, позволяя грамот-

ному программисту проделывать операции, которые не позволяет сам язык, например чтение абсолютного сектора диска. Поэтому материал, относящийся к программированию на среднем уровне, представляет больший интерес, чем может показаться на первый взгляд. Все обсуждения относятся только к операционной системе MS-DOS. Если вы работаете в системе CP/M-86 или UCSD p-system, то вам придется поискать другое руководство.

Наконец, примеры, связанные с программированием на низком уровне, показывают, как данная проблема может быть решена программированием микросхем. Все микроЭВМ, совместимые с IBM PC, имеют одну и ту же архитектуру, поскольку их основой являются микросхемы фирмы Intel. Доступ к микросхемам осуществляется через порты ввода/вывода, к которым вы имеете доступ, работая практически с любым языком, включая Бейсик. Обсуждаются все важные для программиста микросхемы, включая таймер, интерфейс с периферией, контроллер прерываний, контроллер дисплея, контроллер НГМД (накопителя на гибких магнитных дисках) и микросхемы управления коммуникационным каналом. Хотя фирма IBM не рекомендует программировать на этом уровне (из соображений, что такая программа может не работать на последующих модификациях ЭВМ), снова и снова обнаруживаются возможности машины, которые нельзя реализовать другим способом.

Не все задачи показаны на всех трех уровнях. Решение некоторых из них просто невозможно на Бейсике. Для решения других не предусмотрено средств операционной системы. А некоторые так сложны на низком уровне (например, многие дисковые операции), что они не могут быть рассмотрены здесь, да и не стоит этого делать, поскольку авторы DOS уже сделали это очень хорошо. Однако в большинстве случаев показаны все три уровня. Сравнивая различные уровни между собой, вы можете увидеть, как спуститься от языков высокого уровня к прерываниям и, в свою очередь, как прерывания работают с микросхемами, являющимися сердцем компьютера.

Эта книга может показаться ужасной людям, знакомым только с языками высокого уровня, такими, как Бейсик или Паскаль. Это является следствием того, что разделы, относящиеся к программированию на среднем и низком уровнях, написаны на языке ассемблера. Эта книга послужит идеальным компаньоном для тех, кто изучает ассемблер. Но не думайте, что вам нужна только треть книги, если вы не знаете ассемблера и не собираетесь изучать его. Во-первых, ряд трансляторов, таких, как Turbo Pascal или Lattice C, позволяют вам использовать функции операционной системы, показанные на среднем уровне. Кроме того, многие из процедур низкого уровня могут быть на самом деле реализованы на языках высокого уровня. Чтобы вы могли разобраться, что же содержится

в приведенных примерах на ассемблере, в приложении Г дано краткое введение в этот язык. Даже если вы никогда не будете использовать материал, связанный с программированием на низком уровне, внимательный взгляд на него позволит вам намного глубже понять, как же работают языки высокого уровня и почему в некоторых случаях возникают затруднения при работе с ними.

Практически каждый параграф содержит образец кода. Часто это всего лишь несколько тривиальных строк. Иногда приводятся явные наметки для реализации сложных процедур. Очень редко встречаются самостоятельные программы. Вместо того чтобы заполнять книгу изощренными примерами, я в большинстве случаев оставлял лишь фрагмент кода, который понадобится вам, когда вы обратитесь к этой книге за помощью. Приведенные примеры ни в коей мере не претендуют на самое красивое решение проблемы. Основная цель примеров состоит не в том, чтобы предоставить набор готовых программных модулей, а в том, чтобы указать путь решения возникающих проблем, чтобы вы могли начать думать в правильном направлении. Но если вы хотите, то можете прямо включать приведенные образцы в свои программы в качестве функциональной отправной точки и затем дорабатывать их до кондиций, удовлетворяющих вашему эстетическому вкусу. Поскольку все примеры проверены, вы можете спокойно пользоваться ими, чтобы избежать действительно идиотских ошибок, которые имеют тенденцию накапливаться после того, как долгие часы программирования понизят ваш интеллект практически до нулевого IQ.

Язык этой книги, мягко говоря, очень компактный. Но я старался не прибегать к жаргону, насколько это возможно. Кроме того, в конце книги приведен терминологический словарь компьютерных терминов. За исключением материала весьма специального свойства, практически вся относящаяся к программированию информация, доступная из документации IBM, включена в книгу. Хотя, конечно, было бы прекрасно охватить все, но тогда объем книги достиг бы 1000 страниц и за деревьями вы не увидели бы леса. Поэтому для необычных программистских нужд - скажем, для сложных программ управления контроллером НГМД или перепрограммирования клавиатуры AT - вам придется обращаться к техническим руководствам IBM или специальным описаниям, выпускаемым производителями микросхем. Но 99% программ не потребуют другой информации об оборудовании IBM PC, кроме содержащейся в данной книге. Различные способы решения той или иной проблемы собраны в одном месте, и приводится сравнение сильных и слабых сторон того или иного подхода. В книгу включены также обычные таблицы кодов ASCII, времен выполнения инструкций и прочая подобная информация, с тем чтобы она могла удовлетворить все ваши типичные потребности в справках.

Имеется также много информации, которая опущена в документации IBM, например, какие управляющие коды интерпретируются какими программами вывода на экран или как различные дисковые функции работают с файлами. В некоторых параграфах показано решение типичных задач программирования, не связанных напрямую с оборудованием, но использующих некоторую его специфику, таких, как работа в реальном времени или горизонтальная прокрутка. Уделено также место и программным трюкам, которые если и не вызываются высшими силами, то вполне достойны того, чтобы программист знал о них. При существующем положении вещей каждый программист должен открывать эти методы для себя (причем обычно не один раз). Как печально, что высшие жрецы Века Информации тратят так много времени, переизобретая колесо, как в давние времена, когда папирус еще не сделал обмен информацией достаточно легким.

Приводится также информация об отличиях разных версий IBM PC. Все обсуждения относятся к стандартному IBM PC. В тех случаях, когда PCjr, XT или AT ведут себя по-разному, описываются отличительные свойства данной машины. Попутно сразу отметим, что в книге совершенно не рассматриваются сетевые возможности, появившиеся у AT и в MS-DOS 3.0. Эти вопросы заслуживают отдельной книги. За некоторым исключением, все образцы кода рассчитаны на стандартный IBM PC, но пока не сказано обратное, все они будут нормально работать на любом из подвидов. Однако есть существенное ограничение. Все написанное в этой книге предполагает применение MS-DOS 2.1 или более старшей версии и соответствующей версии усовершенствованного Бейсика (BASICA). Пользователям, до сих пор не перешедшим на MS-DOS 2.1, недоступны многие достоинства машины.

Если в этой книге что-то и содержится, то это факты - мириады их, и я искренне надеюсь, что все они верны. В нее включено также несколько сотен примеров программ, и я готов поклясться, что они совершенны. Но если вы думаете, что такое огромное количество информации можно оставить неповрежденным в длительном процессе подготовки книги к изданию, то попробуйте. Если вы обнаружите что-нибудь ужасное, вздохните глубже и подумайте о том, насколько хуже была бы ваша жизнь, если бы этой книги не было. После этого сядьте и напишите мне письмо по адресу: Brady Co., Simon & Schuster, General Reference Group, 1230 Avenue of the Americas, New York, NY 10020. Если вы сделаете это, то жизнь станет немного лучше для тех программистов, которые получат второе издание книги, дополненное сведениями о последних созданиях IBM.

Удачного программирования!

Роберт Джордейн

Соглашения о числах, принятые в этой книге

Программисты на ассемблере не найдут ничего необычного в способе представления чисел и адресов, принятом в этой книге. Но многие программисты, работающие с языками высокого уровня, мало знакомы с системой адресации и недесятичными числами, и они могут быть слегка сконфужены на первых порах. Если вы относитесь к этой категории, не отчаивайтесь! Данная книга сравнительно безболезненно познакомит вас с этой кабалистикой, а ваше образование как программиста будет существенно ограничено без знакомства с этими вещами. Чтобы помочь вам, в книгу включены два приложения. В приложении А обсуждаются двоичные и шестнадцатиричные числа, а также их использование при адресации памяти. В приложении Б более подробно разбираются двоичные числа и их использование в битовых операциях. Даже если вы не нуждаетесь в помощи, обратите внимание на следующие правила:

1. Для удобства менее классных программистов все числа считаются десятичными до тех пор, пока за ними не следует H (для шестнадцатиричных) или B (для двоичных). Иногда B опускается после двоичных чисел, когда очевидно, что их значения описывают цепочку битов.

2. Другое исключение - числа из восьми цифр вида 0000:0000. Это шестнадцатиричные числа, дающие сегмент и смещение адреса памяти. Их значение объяснено в приложении А.

3. Биты нумеруются от 0 до 7 (или от 0 до 15), где бит 0 соответствует младшему биту (т.е. когда установлен бит 0 = 1 и бит 7 = 128).

4. Выражение вида "ASCII 5" относится к символу номер 5 набора ASCII. Это означает, что оно относится к одному байту со значением 5, а не к коду ASCII для символа 5 и не к двухбайтному целому, представляющему значение 5.

5. Числа, заключенные в квадратные скобки, например [2.1.3], являются перекрестными ссылками на другие разделы данной книги. Приведенный пример подразумевает "глава 2, раздел 1, параграф 3". [2.1.0] относится к общему обсуждению, начинающему раздел 1 гл. 2. Вы обнаружите сотни таких ссылок, рассеянных по всему тексту. Они отсылают вас к тем местам книги, где можно найти нужную информацию. Их основное назначение - помощь начинающим. Если вы понимаете, о чем идет речь, игнорируйте перекрестные ссылки.

Глава 1. СИСТЕМНЫЕ РЕСУРСЫ

1.1. Ревизия системных ресурсов

Одной из первых проблем, возникающих после загрузки задачи, является проверка, куда мы "попали": на каком типе IBM PC запущена задача?... под какой версией MS-DOS?... каким объемом памяти мы располагаем?... все ли необходимое оборудование имеется? Существуют три способа получения этой информации. Наименее элегантный из них - спросить об этом у пользователя (но знает ли он, что ответить?). Намного лучше получить всю доступную информацию из установки переключателей на системной плате. Но эта установка не всегда соответствует реальности. Поэтому лучше всего получить прямой доступ к требуемому оборудованию или прочитать нужную информацию из области данных BIOS. Поскольку установка переключателей может служить отправной точкой для получения необходимой информации, то этот раздел начинается с обсуждения микросхемы, содержащей такую информацию, - микросхемы интерфейса с периферией 8255.

Программа может получить доступ к оборудованию только двумя способами. Она может обратиться к любому из портов ввода/вывода, соответствующему присоединенному оборудованию (обычно занята лишь малая доля из 65535 возможных адресов портов), или к любому из более чем миллиона адресов оперативной памяти. Сводная таблица адресов портов приведена в [7.3.0]. На рис. 1.1 показано, как распределены в памяти операционная система и программы.

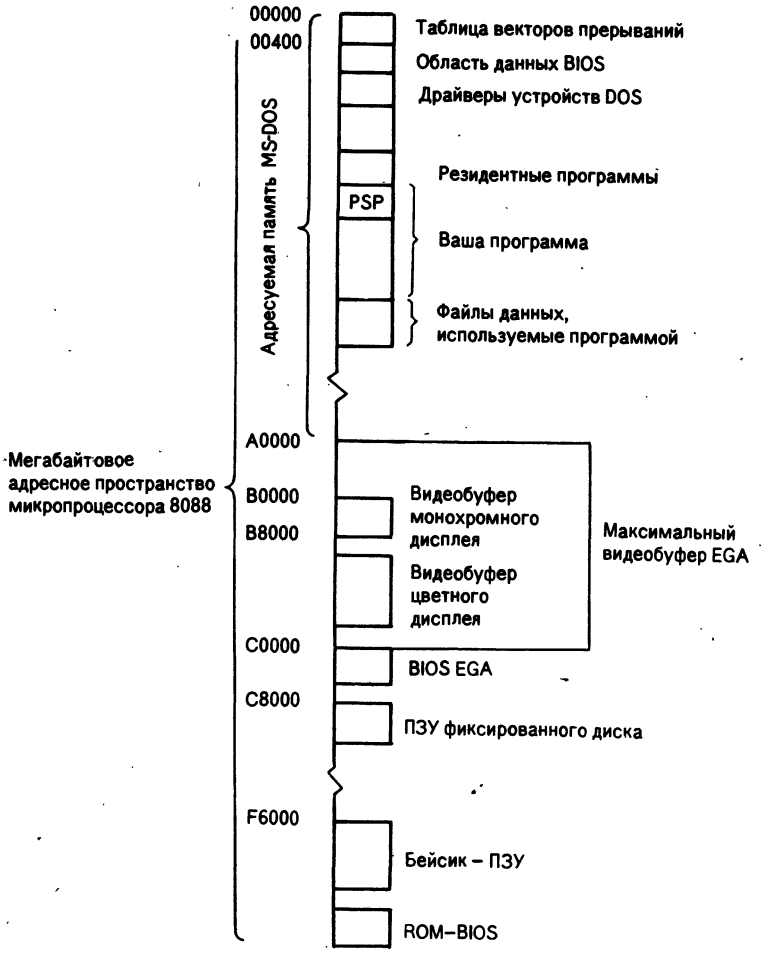


Рис. 1.1. Использование мегабайтового адресного пространства

1.1.1. Доступ к микросхеме интерфейса с периферией 8255

Микросхема интерфейса с периферией Intel 8255 - лучшее место, с которого надо начинать, чтобы получить информацию об имеющемся оборудовании. Эта микросхема предназначена для многих целей. Она сообщает об установке переключателей на

системной плате, принимает для компьютера ввод с клавиатуры, управляет рядом периферийных устройств, включая микросхему таймера 8253. Из машин семейства IBM PC только AT не использует микросхему 8255; он хранит информацию об оборудовании вместе с часами реального времени в специальной микросхеме с независимым питанием. Однако для работы с клавиатурой и управления микросхемой таймера микросхема AT использует те же адреса портов, что и 8255.

Микросхема 8255 имеет три однобайтовых регистра, называемых от порта A до порта C. Адреса этих портов от 60H до 62H соответственно. Считывать данные можно из всех трех портов, но писать можно только в порт B. Для PC установка бита 7 порта B в 1 изменяет информацию, содержащуюся в порте A. Аналогично для PC установка бита 2 определяет содержимое четырех младших битов порта C, а установка бита 3 делает то же самое для XT. Содержимое этих регистров следующее:

Порт A (60H)

когда в порте B бит 7 = 0

биты 0-7 PC,XT,PCjr,AT: 8-битовые скан-коды с клавиатуры

когда в порте B бит 7 = 1 для PC

бит 0 PC: 0 = нет накопителей на дискетах

1 PC: не используется

2-3 PC: число банков памяти на системной плате

4-5 PC: тип дисплея (11 = монохромный,
10 = цветной 80*25, 01 = цветной 40*25)

6-7 PC: число накопителей на дискетах

Порт B (61H)

бит 0 PC,XT,PCjr: управляет каналом 2 таймера 8253

1 PC,XT,PCjr: вывод на динамик

2 PC: выбор содержимого порта C
PCjr: 1 = символьный режим, 0 = графический

3 PC,PCjr: 1 = кассетный мотор выключен
XT: выбор содержимого порта C

4 PC,XT: 0 = разрешение ОЗУ
PCjr: 1 = запрет динамика и мотора кассеты

5 PC,XT: 0 = разрешение ошибок щелей расширения

6 PC,XT: 1 = разрешение часов клавиатуры

5-6 PCjr: выбор динамика (00 = 8253, 01 = кассета,
10 = ввод/вывод, 11 = микросхема 76496)

7 PC: выбор содержимого порта A
PC,XT: подтверждение клавиатуры

Порт C (62H)

когда в порте B бит 2 = 1 для PC или бит 3 = 1 для XT
биты 0-3 PC: нижняя половина переключателя 2 конфигурации (ОЗУ на плате расширения)
0 PCjr: 1 = введенный символ потерян
1 XT: 1 = есть мат. сопроцессор
PCjr: есть карта модема
2 PCjr: есть карта НГМД
2-3 XT: число банков памяти на системной плате
3 PCjr: 0 = 128K памяти
4 PC,PCjr: ввод с кассеты
XT: не используется
5 PC,XT,PCjr: выход канала 2 8253
6 PC,XT: 1 = проверка ошибок щелей расширения
PCjr: 1 = данные с клавиатуры
7 PC,XT: 1 = контроль ошибок четности
PCjr: 0 = кабель клавиатуры подсоединен
когда в порте B бит 2 = 0 для PC или бит 3 = 0 для XT
биты 0-3 PC: верхняя половина переключателя 2 конфигурации (не используется)
0-1 XT: тип дисплея (11 = монохромный,
10 = цветной 80*25, 01 = цветной 40*25)
2-3 XT: число накопителей НГМД (00 = 1 и т.д.)
4-7 PC,XT: то же, что и с установленными битами

Отметим, что 0 в одном из битов регистра соответствует установке переключателя "off".

AT хранит информацию о конфигурации в микросхеме MC146818 фирмы Motorola вместе с часами реального времени. Он вовсе не имеет микросхемы 8255, хотя для управления микросхемой таймера и приема данных с клавиатуры используются те же адреса портов. Микросхема имеет 64 регистра, пронумерованных от 00 до 3FH. Для чтения регистра нужно сначала послать его номер в порт с адресом 70H, а затем прочитать его через порт 71H. Различные параметры конфигурации обсуждаются далее. Приведем здесь только краткую сводку:

Номер регистра	Использование
10H	тип накопителя НГМД
12H	тип накопителя фиксированного диска
14H	периферия
15H	память на системной плате (младший байт)
16H	память на системной плате (старший байт)
17H	общая память (младший байт)
18H	общая память (старший байт)
30H	память сверх 1 мегабайта (младший байт)
31H	память сверх 1 мегабайта (старший байт)

Высокий уровень

В данной книге имеется множество примеров доступа к этим портам. Ниже приводится программа на Бейсике, устанавливающая число дисковых накопителей, присоединенных к IBM PC. Прежде чем прочитать два старших бита порта A, бит 7 порта B должен быть установлен в 1. Вы должны вернуть значение этого бита назад в 0 перед дальнейшей работой, иначе клавиатура будет заперта и для восстановления работоспособности машины вам придется выключить ее. Бейсик не позволяет двоичное представление чисел, что затрудняет работу с цепочками битов. Простая подпрограмма может заменить любое целое число вплоть до 255 (максимальное значение, которое может принимать номер порта) на восьмисимвольную двоичную строку. После этого строковая функция MID$ позволяет вырезать нужные биты для анализа. Основы битовых операций в Бейсике описаны в приложении Б.

```
100 A = INP(&H61)                    'получаем значение из порта B
110 A = A OR 128                     'устанавливаем бит 7
120 OUT &H61,A                       'посылаем байт назад в порт B
130 B = INP(&H60)                    'получаем значение из порта A
140 A = A AND 128                    'сбрасываем бит 7
150 OUT &H61,A                       'восстанавливаем значение порта B
160 GOSUB 1000                       'преобразуем в двоичную строку
170 NUMDISK$ = RIGHT$(B$,1)          'получаем нулевой бит
180 IF D$ = 1 THEN NUMDISK = 0: GOTO 230 'нет дисков
190 C$ = LEFT$(B$,2)                 'берем два старших бита строки
```

```
200 TALLEY = 0                          'переменная для числа дисков
210 IF RIGHT$(C$,1) = "1" THEN TALLEY = 2 'берем старший бит
220 IF LEFT$(C$,1) = "1" THEN TALLEY = TALLEY + 1 'и младший
230 TALLEY = TALLEY + 1                 'счет начинается с 1, а не с 0
'теперь имеем число накопителей
1000 '''Подпрограмма преобразования байта в двоичную строку
1010 B$ = ""                            'заводим строку
1020 FOR N = 7 TO 0 STEP -1             'проверка очередной степени 2
1030 Z = B - 2^N
1040 IF Z >= 0 THEN B = Z: B$ = B$ + "1" ELSE B$ = B$ + "0"
1050 NEXT                               'повторяем для каждого бита
1060 RETURN                             'все закончено
```

Низкий уровень

Ассемблерная программа получает число имеющихся дисковых накопителей тем же способом, что и в приведенном выше примере, но более просто. Напоминаем, что нельзя забывать о восстановлении первоначального значения в порте B.

```
IN    AL,61H          ;получаем значение из порта B
OR    AL,10000000B    ;устанавливаем бит 7 в 1
OUT   61H,AL          ;заменяем байт
IN    AL,60H          ;получаем значение из порта A
MOV   CL,6            ;подготовка для сдвига AL
SHR   AL,CL           ;сдвигаем 2 старших бита на 6 позиций
INC   AL              ;начинаем счет с 1, а не с 0
MOV   NUM_DRIVES,AL   ;получаем число накопителей
IN    AL,61H          ;подготовка к восстановлению порта B
AND   AL,01111111B    ;сбрасываем бит 7
OUT   61H,AL          ;восстанавливаем байт
```

1.1.2. Определение типа IBM PC

Существуют проблемы совместимости между различными типами IBM PC. Для того чтобы программа могла работать на любом из IBM PC, используя все его возможности, необходимо, чтобы она могла определить тип машины, в которую она загружена. Эта информация содержится во втором с конца байте памяти по адресу FFFFE в ROM-BIOS. Ключевые числа следующие:

Компьютер	Код
PC	FF
XT	FE
PCjr	FD
AT	FC

Высокий уровень

В Бейсике надо просто использовать PEEK для чтения значения:

```
100 DEF SEG = &HF000          'указываем на верхние 64K памяти
110 X = PEEK(&HFFFE)          'читаем второй с конца байт
120 IF X = &HFD THEN ...      '... тогда это PCjr
```

Низкий уровень

В языке ассемблера:

```
;--- определение типа компьютера:
MOV  AX,0F000H          ;указывает ES на ПЗУ
MOV  ES,AX
MOV  AL,ES:[0FFFEH]     ;получаем байт
CMP  AL,0FDH            ;это PCjr?
JE   INITIALIZE_JR      ;переходим на инициализацию
```

1.1.3. Определение версии MS-DOS

По мере развития MS-DOS ее возможности расширялись. Это существенно облегчило написание определенных частей программы по сравнению с предыдущими версиями. Чтобы иметь гарантию, что программа будет работать с любой версией MS-DOS, она должна использовать только функции, доступные в MS-DOS 1.0. В системе предусмотрено прерывание, возвращающее номер версии MS-DOS. Это число может служить для проверки выполнимости вашей программы. При старте программа может выдавать сообщение, что ей нужна другая версия MS-DOS.

Средний уровень

Функция 30H прерывания 21H возвращает номер версии MS-DOS. Старший номер версии (2 из 2.10) возвращается в AL, а

младший номер версии (10 из 2.10) возвращается в AH (обратите внимание, что младший номер .1 возвращает значение AH, а не 1H). AL может содержать 0, что указывает на версию MS-DOS, более раннюю, чем 2.0. Это прерывание меняет содержимое регистров BX и CX, в которых возвращается значение 0.

```
;--- определение версии MS-DOS:
MOV  AH,30H          ;номер функции получения версии
INT  21H             ;получить номер версии
CMP  AL,2            ;проверка на версию 2.x
JL   WRONG_DOS       ;если меньше 2, то выдать сообщение
```

1.1.4. Определение числа и типов адаптеров дисплея

Программе может оказаться необходима информация о том, будет ли она работать в системе с монохромным адаптером, с цветной графической картой или с EGA, а также информация о наличии второго адаптера. В параграфе [4.1.6] объяснено, как передать управление от одного адаптера к другому. Байт статуса оборудования, хранящийся в области данных BIOS по адресу 0040:0010, сообщает установку переключателя 1, который показывает, какая из карт активна. В принципе биты 5-4 должны иметь значение 11 для монохромной карты, 10 - для цветной карты 80*25, 01 - для цветной карты 40*25 и 00 - для EGA. Однако при наличии EGA он может установить значение битов отличным от 00, в зависимости от установки его собственных переключателей. Поэтому вы должны сначала с помощью других средств определить наличие EGA, а затем, если его нет, по данным BIOS определить, является ли активным цветной или монохромный адаптер. Для проверки наличия EGA надо прочитать байт по адресу 0040:0087. Если он равен 0, то EGA отсутствует. Если этот байт не нулевой, то, когда бит 3 = 0, EGA является активным адаптером, а когда он равен 1, то активен второй адаптер.

Когда присутствует EGA, проверка наличия монохромного или цветного адаптера осуществляется записью значения в регистр адреса курсора микросхемы 6845 [4.1.1] и последующего чтения и проверки значений на совпадение. Для монохромной карты пошлите 0FH в порт 3B4H, чтобы указать на регистр курсора, а затем прочитайте и запишите адрес курсора через порт 3B5H. Соответствующие порты для цветной карты - 3D4H и 3D5H. Когда карта отсутствует, порт возвращает значение 0FFH; но поскольку это значение может содержаться в регистре, простой проверки на него недостаточно.

При наличии EGA могут потребоваться ответы на следующие два вопроса: сколько имеется памяти на его карте и какой тип монитора подсоединен? Для определения типа дисплея проверьте бит 1 по адресу 0040:0087; когда он установлен, подсоединен монохромный дисплей, а когда он равен нулю - цветной. Если ваша программа использует цветной графический режим с 350 строками, то надо также выяснить, какой дисплей присоединен: IRGB или R'G'B'RGB, где последняя аббревиатура соответствует улучшенному цветному дисплею IBM. Это определяется установкой четырех переключателей на карте EGA. Установка этих переключателей возвращается в CL при обращении к функции 12H прерывания 10H. Цепочка четырех младших битов должна быть 0110 для улучшенного цветного дисплея. Та же функция сообщает и о наличии памяти на карте EGA. Она возвращает BL, содержащий 0 для 64K, 1 для 128K, 2 для 192K и 3 для полных 256K памяти дисплея.

Высокий уровень

Приведенные фрагменты кода определяют тип текущего монитора и режим его работы, а также типы видеоадаптеров в машине:

```
100 ''определение активного адаптера
110 DEF SEG = &H40                  'указываем на область данных BIOS
120 X = PEEK(&H87)                  'проверка на наличие EGA
130 IF X = 0 THEN 200               'EGA отсутствует, идем дальше
140 IF X AND 8 = 0 THEN...          'активный монитор EGA
.
.
200 X = PEEK(&H10)                  'читаем байт статуса оборудования
210 Y = X AND 48                    'выделяем биты 4 и 5
220 IF Y = 48 THEN ...              '... тогда монохромный (00110000)
230 IF Y = 32 THEN ...              '... тогда цветной 80*25 (00100000)
240 IF Y = 16 THEN ...              '... тогда цветной 40*25 (00010000)
```

Следующий пример иллюстрирует проверку наличия монохромной карты, когда активной является карта EGA или цветная. Он может служить и для проверки наличия цветной карты, если использовать адреса портов &H3D4 и &H3D5.

```
100 '''проверка наличия монохромной карты
110 OUT &H3B4,&HF                адрес регистра курсора
120 X = INP(&H3B5)               'чтение и сохранение значения
130 OUT &H3B5,100                'посылаем в регистр любое значение
140 IF INP(&H3B5)<>100 THEN...   'если карта есть - вернется то же
150 OUT &H3B5,X                  'восстанавливаем значение регистра
```

Низкий уровень

Предлагаемые ниже примеры соответствуют примерам на Бейсике, приведенным ранее:

```
;--- определение активного адаптера:
            MOV  AX,40H          ;указываем ES на область данных BIOS
            MOV  ES,AX
            MOV  AL,ES:[87H]     ;проверяем наличие EGA
            CMP  AL,0
            JE   NO_EGA          ;если 0040:0087 = 0, то EGA нет
            TEST AL,00001000B    ;EGA есть, проверяем бит 3
            JNZ  EGA_NA          ;если бит 3 = 1, то EGA неактивен
            .
            .
            .
EGA_NA:     MOV  AL,ES:[1011]    ;проверяем байт статуса дисплея
            AND  AL,00110000B    ;выделяем биты 4 и 5
            CMP  AL,48           ;это монохромная карта?
            JE   MONOCHROME      ;переход,если да
```

Предполагая наличие монохромной карты, проверим, установлена ли цветная карта (неактивная):

```
;--- установлена ли неактивная цветная карта?
MOV  DX,3D4H          ;указываем на регистр адреса 6845
MOV  AL,0FH           ;запрашиваем регистр курсора
OUT  DX,AL            ;указываем на регистр
INC  DX               ;указываем на регистр данных
IN   AL,DX            ;получаем текущее значение
XCNG AH,AL            ;сохраняем значение
MOV  AL,100           ;тестовое значение 100
OUT  DX,AL            ;посылаем его
```

```
IN    AL,DX        ;считываем его снова
CMP   AL,100       ;сравниваем значения
JNE   NO_CARD      ;переход, если нет карты
XCNG  AH,AL        ;иначе, есть цветная карта
OUT   DX,AL        ;тогда восстанавливаем значение
```

1.1.5. Определение числа и типа дисковых накопителей

На всех машинах, кроме AT (который будет обсуждаться ниже), регистры микросхемы 8255 интерфейса с периферией содержат информацию о том, сколько НГМД имеет машина. В примерах из [1.1.1] показано, как получить эту информацию. Информация, определяющая тип диска, находится в таблице размещения файлов (FAT) диска, которая следит за использованием дискового пространства. Первым байтом в FAT может быть один из следующих кодов:

Код	Тип диска
FF	двухсторонний, 8 секторов
FE	односторонний, 8 секторов
FD	двухсторонний, 9 секторов
FC	односторонний, 9 секторов
F9	двухсторонний, 15 секторов
F8	фиксированный диск

Таблица размещения файлов не является файлом. Она может быть считана при помощи функций DOS или BIOS, непосредственно читающих определенные секторы диска. В параграфе [5.1.1] содержится вся информация, необходимая для нахождения и чтения FAT. К счастью, операционная система обеспечивает функцию, которая возвращает идентификационный байт диска.

Данные BIOS не показывают число жестких дисков в системе, так как переключатели предназначены только для гибких дисков. Однако вы можете использовать указанную функцию операционной системы для поиска накопителей. Она возвращает значение 0CDH вместо одного из упомянутых кодов, когда накопители отсутствуют. Надо просто проверять все бо́льшие и бо́льшие номера накопителей до тех пор, пока не будет обнаружено указанное значение.

AT уникален в том смысле, что его информация о конфигурации говорит, какой тип накопителя используется. Эту информацию можно получить из порта с адресом 71H, предварительно послав номер регистра в порт 70H. Для НГМД номер регистра равен 10H. Информация о первом накопителе содержится в битах 7-4, а о втором - в битах 3-0. В обоих случаях цепочка битов 0000 говорит об отсутствии накопителя, 0001 - о двухстороннем накопителе с плотностью 48 дорожек на дюйм, а 0010 - о накопителе большой емкости (96 дорожек на дюйм). Информация о фиксированном диске содержится в регистре 12H. И снова биты 7-4 и 3-0 соответствуют первому и второму накопителям. 0000 указывает на отсутствие накопителя. Другие 15 возможных значений описывают емкость и конструкцию накопителя. Эти коды сложные; если вам по какой-то причине потребуется эта информация, обратитесь к техническому руководству по AT.

Средний уровень

Функция 1CH прерывания 21H возвращает информацию об указанном накопителе. Поместите номер накопителя в DL, причем 0 = накопитель по умолчанию, 1 = A и т.д. При возвращении DX содержит число кластеров в FAT, AL - число секторов в кластере, а CX - число байтов в секторе. DS:BX указывает на байт, содержащий код идентификации диска из FAT, согласно приведенной таблице. В следующем примере определяется тип накопителя A:

```
;---определение типа диска
MOV  AH,1CH          ;функция MS-DOS
MOV  DL,1            ;выбор накопителя A
INT  21H             ;получение информации
MOV  DL,[BX]         ;получение типа накопителя
CMP  DL,0FDH         ;двухсторонний, 9 секторов?
JE   DBL_9           ;и т.д.
```

BIOS AT имеет функцию, сообщающую общие параметры накопителей. Это функция 8 прерывания 13H. Она возвращает число накопителей в DL, максимальное число сторон накопителя - в DH, максимальное число секторов - в CL и дорожек - в CH, а код статуса ошибки накопителя - в AH (см. параграф [5.4.8]).

Другая функция BIOS AT возвращает тип накопителя. Это функция 15H прерывания 13H, которая требует номера накопителя в DL. В AH возвращается код, причем 0 = нет

накопителя, 1 = дискета без обнаружения изменений, 2 = дискета с обнаружением изменений и 3 = фиксированный диск. В случае фиксированного диска в CX:DX возвращается число секторов по 512 байт.

1.1.6. Определение числа и типа периферийных устройств

При старте ROM-BIOS проверяет присоединенное оборудование, сообщая о результатах своей проверки в регистр статуса. Этот регистр занимает два байта, начиная с 0040:0010. Приведенные ниже значения битов относятся ко всем машинам, пока не оговорено обратное:

бит 0	если 1, то присутствует НГМД
1	XT,AT:1 = есть математический сопроцессор
	(PC,PCjr:не используется)
2-3	11 = базовая память 64K (AT:не используется)
4-5	Активный видеоадаптер (11 = монохромный,
	10 = цветной 80*25, 01 = цветной 40*25)
6-7	число НГМД (если бит 0 = 1)
8	PCjr:0 = есть DMA (PC,XT,AT:не используется)
9-11	число адаптеров коммуникации
12	1 = есть игровой порт (AT:не используется)
13	PCjr:есть серийный принтер (PC,XT,AT:не используется)
14-15	число присоединенных принтеров

Большая часть информации расшифровывается примитивно. Но обратите внимание, что информация о дисковых накопителях распределена между битами 0 и 6-7. Значение 0 в битах 6-7 указывает, что имеется один дисковый накопитель; чтобы узнать об отсутствии накопителей, надо проверить бит 0.

Число портов коммуникации может быть получено из области данных BIOS. BIOS отводит четыре 2-байтовых поля для хранения базовых адресов вплоть до четырех COM портов (MS-DOS использует только два из них). Базовый адрес - это младший из адресов портов, относящихся к группе портов, имеющих доступ к данному каналу коммуникации. Эти четыре поля начинаются с адреса 0040:0008. Порту COM1 соответствует адрес :0008, а COM2 - 000A. Если это поле содержит 0, то соответствующий порт отсутствует. Таким образом, если слово по адресу :0008 отлично от нуля, а по адресу 000A - нулевое, то имеется один порт коммуникации.

AT хранит информацию о периферии в регистре 14H микросхемы конфигурации. Сначала запишите 14H в порт с адресом 70H, а затем прочитайте содержимое регистра через порт 71H. Вот значение битов этого регистра:

биты	7-6	00 = 1 НГМД, 01 = 2 НГМД
	5-4	01 = вывод на цветной дисплей, 40 строк
		10 = вывод на цветной дисплей, 80 строк
		11 = вывод на монохромный дисплей
	3-2	не используется
	1	1 = имеется математический сопроцессор
	0	0 = нет НГМД, 1 = имеется НГМД

Высокий уровень

В Бейсике нужно просто прочитать байты статуса из области данных BIOS. В приложении Б объяснено выполнение битовых операций в Бейсике. В приведенном примере проверка наличия дисковых накопителей достигается проверкой четности младшего байта статусного регистра (четный - нет накопителей).

```
100 DEF SEG = 0                    'указываем на дно памяти
110 X = PEEK(&H410)                'получаем младший байт регистра
120 IF X MOD 2 = 0 THEN 140        'он четный - нет накопителей
130 PRINT "Имеется диск"           'иначе - имеется накопитель
140 GOTO 160                       'идем ко второму сообщению
150 PRINT "Нет накопителей"        'второе сообщение
160 ...                            'продолжаем...
```

Проверка наличия COM1:

```
100 DEF SEG = 40H                  'указываем на область данных BIOS
110 PORT = PEEK(0) + 256*PEEK(1)   'получаем слово со смещением 0
120 IF PORT = 0 THEN...            '... то нет адаптера COM1
```

Средний уровень

Прерывание 11H BIOS возвращает байт статуса оборудования в AX. На входе ничего подавать не надо. В примере определяется число дисковых накопителей.

```
;---получение числа дисковых накопителей:
INT   11H              ;получаем байт статуса
TEST  AL,0             ;имеются накопители?
JZ    NO_DRIVES        ;переход, если нет
AND   AL,1100000B      ;выделяем биты 5
MOV   CL,5             ;подготовка к сдвигу регистра
SHR   AL,CL            ;сдвиг вправо на 5 бит
INC   AL               ;добавляем 1, так как отсчет идет с 1
```

Низкий уровень

Ассемблерная программа работает так же, как и программа на Бейсике. Пример иллюстрирует чтение информации о конфигурации для AT и определение, установлен ли математический сопроцессор:

```
MOV   AL,14H           ;номер регистра
OUT   70H,AL           ;посылаем запрос
IN    AL,71H           ;читаем регистр
TEST  AL,10B           ;проверяем бит 1
JZ    NO_COPROC        ;если не установлен, то сопроцессора нет
```

1.1.7. Ревизия количества памяти

Вопрос: "Сколько имеется памяти?" - может быть задан в трех смыслах. О каком количестве памяти сообщают переключатели, установленные на системной плате? Сколько микросхем памяти реально установлено в машине? И наконец, сколько остается свободной памяти, которую DOS может использовать для выполнения ваших программ? Машина может иметь 10 банков памяти по 64K, но переключатели способны указывать на наличие только 320K, оставляя половину памяти для каких-либо специальных целей. А как ваша программа узнает, сколько из доступных 320K она может использовать, учитывая, что другое программное обеспечение может быть загружено резидентным в верхнюю или нижнюю часть памяти?

Ответ на каждый вопрос можно получить определенным способом. Для PC и XT установка переключателей может быть просто прочитана через порт B микросхемы интерфейса с периферией 8255. В параграфе [1.1.1] описано, как это делается. BIOS хранит двухбайтовую переменную по адресу 0040:0013, которая сообщает число килобайтов используемой памяти. Для PCjr бит 3 порта 62H (порт C микросхемы 8255) равен нулю,

когда машина имеет добавочные 64K памяти. AT дает особо полную информацию о памяти. Регистры 15H (младший) и 16H (старший) микросхемы информации о конфигурации сообщают, сколько памяти установлено на системной плате (возможны три значения: 0100H для 256K, 0200H для 512K и 0280H для 512K плюс 128K на плате расширения). Память канала ввода/вывода для AT сообщается регистрами 17H и 18H (с инкрементом 512K). Память сверх 1 мегабайта доступна через регистры 30H и 31H (с инкрементом 512K вплоть до 15 мегабайт). Если AT имеет 128K на плате расширения, то установлен бит 7 регистра 33. Во всех случаях надо сначала послать номер регистра в порт 70H, а затем прочитать значение из порта 71H.

Легко написать программу, которая прямо тестирует наличие памяти через определенные интервалы адресного пространства. Поскольку минимальная порция памяти - 16 килобайт, достаточно проверить одну ячейку памяти в каждом 16-килобайтовом сегменте, чтобы убедиться, что все 16K присутствуют. Когда данная ячейка памяти отсутствует, при чтении из нее получаем значение 233. Для проверки можно записать в ячейку произвольное число, отличное от 233, и сразу же считать его. Если вместо посланного числа возвращается 233, то соответствующий банк памяти отсутствует. Не применяйте этот способ на AT, где при попытке писать в несуществующую память вступает в действие встроенная обработка несуществующей памяти. Диагностика AT настолько хороша, что вы можете целиком положиться на системную информацию о конфигурации.

Память постоянно занимается частями операционной системы, драйверами устройств, резидентными программами обработки прерываний и управляющими блоками MS-DOS. При проверке банков памяти вы не должны вносить необратимых изменений в содержимое памяти. Сначала надо сохранить значение, хранящееся в тестируемой ячейке, затем проверить ее и восстановить первоначальное значение.

Имеется еще одна проблема. Если ваша процедура хотя бы временно модифицирует свой код, то это может привести к сбою. Поэтому для проверки надо выбирать такую ячейку из блока 64K, которая не будет занята текстом вашей процедуры. Для этого поместите процедуру тестирования впереди программы, а для тестирования выберите ячейку со смещением, равным смещению для кодового сегмента. Например, если регистр кодового сегмента содержит 13E2, то сегмент начинается со смещения 3E2 во втором 64-килобайтовом блоке памяти. Поскольку ваша подпрограмма проверки не может находиться по этому адресу, вы можете безопасно проверять ячейку 3E2 в каждом блоке. Запрет прерываний (см. [1.2.2]) позволяет не беспокоиться о

модификации кода из-за аппаратных прерываний, которые могут происходить во время проверки.

Определение количества памяти, реально доступной операционной системе, также требует некоторого фокуса. Когда программа первый раз получает управление, DOS отводит ей всю доступную память, включая верхнюю область, содержащую нерезидентную часть DOS (которая автоматически перезагружается, если она была модифицирована). Для запуска другой программы из текущей или для того, чтобы сделать программу подходящей для многопользовательской системы, необходимо урезать программу до требуемого размера. В параграфе [1.3.1] описано, как это сделать с помощью функции 4AH прерывания 21H.

Эта же функция может быть использована для расширения отведенной памяти. Поскольку программе отводится вся доступная память при загрузке, такое расширение невозможно при старте. Если вы попробуете сделать это, то будет установлен флаг переноса, в регистре AX появится код ошибки 8, а в регистре BX будет возвращено максимальное число доступных 16-байтовых параграфов. Эта информация как раз и нужна. Значит, надо выдать запрос со слишком большим значением в регистре BX (скажем, F000H параграфов), а затем выполнить прерывание. Позаботьтесь о том, чтобы выполнить эту функцию в самом начале программы, пока регистр ES еще имеет начальное значение.

Высокий уровень

Интерпретатор Бейсика использует только 64K (хотя операторы PEEK и POKE позволяют доступ к памяти за пределами 64K). Доля памяти, доступная в настоящий момент, возвращается функцией FRE. Эта функция имеет фиктивный аргумент, который может быть числовым или символьной строкой. BYTES = FRE(x) передает в BYTES число свободных байтов. BYTES = FRE(x$) делает то же самое. Но строковый аргумент вынуждает очистку области данных перед тем, как возвратить число байтов. Заметим, что если размер рабочей области устанавливается с помощью оператора CLEAR, то количество памяти, сообщаемое функцией FRE, будет на 2,5 - 4 килобайта меньше из-за потребностей рабочей области интерпретатора.

Транслятор Бейсика не накладывает ограничение 64K на суммарный объем кода и данных. Но компилятор ограничен тем количеством памяти, которое он может использовать при компиляции. Если этого пространства недостаточно, то уничтожьте

все ненужные номера строк при помощи ключа компиляции /N. Можно также использовать более короткие имена переменных.

Средний уровень

Прерывание 12H BIOS проверяет установку переключателей и возвращает в AX количество килобайт памяти в системе. Эта величина вычисляется из установки регистров микросхемы 8255 или, для AT, микросхемы конфигурации/часов. Входных регистров нет. Имейте в виду, что установка переключателей может быть неверной и это ограничивает достоверность такого подхода.

Для определения числа 16-байтовых параграфов, доступных для DOS, используйте функцию 4AH прерывания 21H. ES должен иметь то же значение, что при старте задачи:

```
;---определение числа параграфов,доступных для DOS
MOV  AH,4AH          ;указываем нужную функцию
MOV  BX,0FFFFH       ;требуем слишком большую память
INT  21H             ;BX содержит число доступных параграфов
```

AT использует функцию 88H прерывания 15H для проверки наличия расширенной памяти. Эта функция ищет память вне адресного пространства процессора в обычном режиме адресации. Говорят, что она ищет память за отметкой 1 мегабайт. Чтобы эта функция работала, на системной плате должно быть от 512 до 640 килобайт памяти. Число килобайтовых блоков расширенной памяти возвращается в AX.

Низкий уровень

В первом примере приведена проверка числа банков памяти по 64K в первых десяти 64-килобайтовых сегментах памяти. Если вы будете проверять старшие 6 банков памяти, то имейте в виду, что имеются видеобуфер, начиная с B000:0000 (и, возможно, A000:0000), и ПЗУ, начиная с F000:0000 (и, возможно, C000:0000).

```
;---проверка каждого банка памяти:
CLI                  ;запрет аппаратных прерываний
MOV  AX,CS           ;получаем значение кодового сегмента
AND  AX,0FFFH        ;сбрасываем старшие 4 бита
MOV  ES,AX           ;помещаем указатель в ES
```

```
MOV  DI,0          ;DI считает число банков памяти
MOV  CX,10         ;будем проверять 10 банков
MOV  BL,'X'        ;для проверки используем 'X' NEXT:
MOV  DL,ES:[0]     ;сохраняем значение тестируемой ячейки
MOV  ES:[0],BL     ;помещаем 'X' в эту ячейку
MOV  DH,ES:[0]     ;читаем тестируемую ячейку
MOV  ES:[0],DL     ;восстанавливаем значение
CMP  DH,'X'        ;совпадает с тем, что писали?
JNE  GO_AHEAD      ;если нет, то банк отсутствует
INC  DI            ;увеличиваем число банков GO_AHEAD:
MOV  AX,ES         ;готовим увеличение указателя
ADD  AX,1000H      ;указываем на следующие 64K
MOV  ES,AX         ;возвращаем указатель в ES
LOOP NEXT          ;обрабатываем следующий банк
STI                ;разрешаем аппаратные прерывания
```

1.2. Управление прерываниями

Прерывания - это готовые процедуры, которые компьютер вызывает для выполнения определенной задачи. Существуют аппаратные и программные прерывания. Аппаратные прерывания инициируются аппаратурой, либо с системной платы, либо с карты расширения. Они могут быть вызваны сигналом микросхемы таймера, сигналом от принтера, нажатием клавиши на клавиатуре и множеством других причин. Аппаратные прерывания не координируются с работой программного обеспечения. Когда вызывается прерывание, процессор оставляет свою работу, выполняет прерывание, а затем возвращается на прежнее место. Для того чтобы иметь возможность вернуться точно в нужное место программы, адрес этого места (CS:IP) запоминается на стеке вместе с регистром флагов. Затем в CS:IP загружается адрес программы обработки прерывания и ей передается управление. Программы обработки прерываний иногда называют драйверами прерываний. Они всегда заканчиваются инструкцией IRET (возврат из прерывания), которая завершает процесс, начатый прерыванием, возвращая старые значения CS:IP и регистра флагов, давая тем самым программе возможность продолжить выполнение из того же состояния.

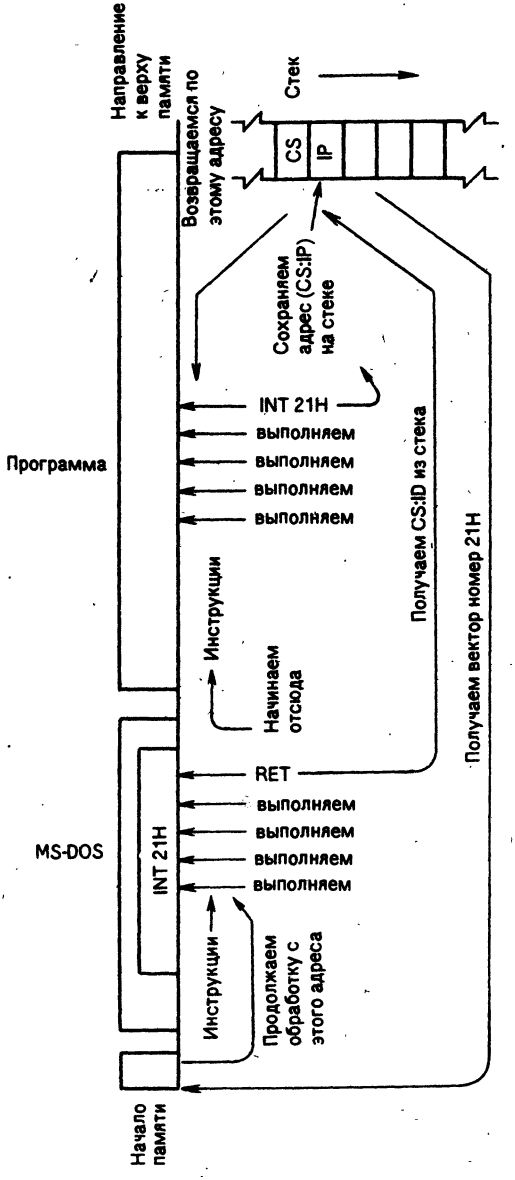


Рис. 1.2. Выполнение прерывания 21H

С другой стороны, программные прерывания на самом деле ничего не прерывают. Это обычные процедуры, которые вызываются вашими программами для выполнения рутинной работы, такой, как обработка нажатия клавиши на клавиатуре или вывод на экран. Однако эти подпрограммы содержатся не внутри вашей программы, а в операционной системе, и механизм прерываний дает вам возможность обратиться к ним. Программные прерывания могут вызываться друг из друга. Например, все прерывания обработки ввода с клавиатуры DOS используют прерывания обработки ввода с клавиатуры BIOS для получения символа из буфера клавиатуры. Отметим, что аппаратное прерывание может получить управление при выполнении программного прерывания. При этом не возникает конфликтов, так как каждая подпрограмма обработки прерывания сохраняет значения всех используемых ею регистров и затем восстанавливает их при выходе, тем самым не оставляя следов того, что она занимала процессор.

Адреса программ прерываний называют *векторами*. Каждый вектор имеет длину 4 байта. В первом слове хранится значение IP, а во втором - CS. Младшие 1024 байта памяти содержат векторы прерываний, таким образом, есть место для 256 векторов. Вместе взятые они называются таблицей векторов. Вектор для прерывания 0 начинается с ячейки 0000:0000, для прерывания 1 - с ячейки 0000:0004, 2 - с 0000:0008 и т.д. Если вы посмотрите на 4 байта, начиная с адреса 0000:0020, в которых содержится вектор прерывания 8H (прерывание времени суток), то вы обнаружите там A5FE00F0. Имея в виду, что младший байт слова расположен сначала и что порядок - IP:CS, это 4-байтовое значение переводится в F000:FEA5. Это стартовый адрес программы ПЗУ, выполняющей прерывание 8H. На рис. 1.2 показана схема выполнения программой прерывания 21H.

1.2.1. Программирование контроллера прерываний 8259

Для управления аппаратными прерываниями во всех типах IBM PC используется микросхема программируемого контроллера прерываний Intel 8259. Поскольку в каждый момент времени может поступить не один запрос, микросхема имеет схему приоритетов. Имеется 8 уровней приоритетов, кроме AT, у которого их 16, и обращения к соответствующим уровням обозначаются сокращениями от IRQ0 до IRQ7 (от IRQ0 до IRQ15), что означает запрос на прерывание. Максимальный приоритет соответствует уровню 0. Добавочные 8 уровней для AT обрабатываются второй микросхемой 8259; этот второй набор уровней имеет приоритет между IRQ2 и IRQ3. Запросы на

прерывание 0-7 соответствуют векторам прерываний от 8H до 0FH; для AT запросы на прерывания 8-15 обслуживаются векторами от 70H до 77H. Ниже приведены назначения этих прерываний.

		Аппаратные прерывания в порядке приоритета
IRQ 0		таймер
1		клавиатура
2		канал ввода/вывода
	8	часы реального времени (только AT)
	9	программно переводятся в IRQ2 (только AT)
	10	резерв
	11	резерв
	12	резерв
	13	математический сопроцессор (только AT)
	14	контроллер фиксированного диска (только AT)
	15	резерв
3		COM1 (COM2 для AT)
4		COM2 (модем для PCjr, COM1 для AT)
5		фиксированный диск (LPT2 для AT)
6		контроллер дискет
7		LPT1

Прерыванию времени суток [2.1.0] дан максимальный приоритет, поскольку если оно будет постоянно теряться, то будут неверными показания системных часов. Прерывание от клавиатуры [3.1.0] вызывается при нажатии или отпускании клавиши; оно вызывает цепь событий, которая обычно заканчивается тем, что код клавиши помещается в буфер клавиатуры (откуда он затем может быть получен программными прерываниями).

Микросхема 8259 имеет три однобайтовых регистра, которые управляют восемью линиями аппаратных прерываний. Регистр запроса на прерывание (IRR) устанавливает соответствующий бит, когда линия прерывания сигнализирует о запросе. Затем микросхема автоматически проверяет, не обрабатывается ли другое прерывание. При этом она запрашивает информацию регистра обслуживания (ISR). Дополнительная цепь отвечает за схему приоритетов. Наконец, перед вызовом прерывания проверяется регистр маски прерываний (IMR), чтобы узнать, разрешено ли в данный момент прерывание данного уровня. Как

правило, программисты обращаются только к регистру маски прерываний через порт 21H [1.2.2] и командному регистру прерываний через порт 20H [1.2.3].

1.2.2. Запрет/разрешение отдельных аппаратных прерываний

Программы на ассемблере могут запретить аппаратные прерывания, перечисленные в [1.2.1]. Это маскируемые прерывания; другие аппаратные прерывания, возникающие при некоторых ошибках (таких, как деление на ноль), не могут быть маскированы. Существуют две причины для запрета аппаратных прерываний. В первом случае все прерывания блокируются, с тем чтобы критическая часть кода была выполнена целиком, прежде чем машина произведет какое-либо другое действие. Например, прерывания запрещают при изменении вектора аппаратного прерывания, чтобы избежать выполнения прерывания, когда вектор изменен только наполовину.

Во втором случае маскируются только определенные аппаратные прерывания. Это делается, когда некоторые прерывания могут взаимодействовать с операциями, критичными к временным интервалам. Например, точно рассчитанная по времени процедура ввода/вывода не может себе позволить быть прерванной длительным дисковым прерыванием.

Низкий уровень

Выполнение прерываний зависит от значения флага прерывания (бит 9) в регистре флагов. Когда этот бит равен 0, разрешены все прерывания, которые разрешает маска. Когда он равен 1, все аппаратные прерывания запрещены. Чтобы запретить прерывания, установив этот флаг в 1, используется инструкция CLI. Для очистки этого флага и восстановления прерываний применяется инструкция STI. Избегайте отключения прерываний на длительный период. Прерывание времени суток происходит 18,2 раза в секунду, и если к этому прерыванию был более чем один запрос в то время, когда аппаратные прерывания были запрещены, то лишние запросы будут отброшены и системное время будет определяться неправильно.

Имейте в виду, что машина автоматически запрещает аппаратные прерывания при вызове программных прерываний и автоматически разрешает их при возврате. Когда вы пишете свои программные прерывания, начинайте программу с инструкции STI, если можно допустить аппаратные прерывания. Отметим также, что

если за инструкцией CLI не следует STI, то это приведет к остановке машины, так как ввод с клавиатуры будет заморожен.

Для маскирования определенных аппаратных прерываний нужно просто послать требуемую цепочку битов в порт с адресом 21H, который соответствует регистру маски прерываний (IMR). Регистр маски на второй микросхеме 8259 для AT (IRQ8-15) имеет адрес порта A1H. Установите те биты регистра, которые соответствуют номерам прерываний, выбранным вами для маскирования. Этот регистр можно только записывать. Приведенный ниже пример блокирует дисковое прерывание. Не забудьте очистить регистр в конце программы, иначе обращение к дискам будет запрещено и после завершения программы.

```
;---маскирование 6-го бита регистра маски прерываний
MOV  AL,01000000B      ;маскируем бит 6
OUT  21H,AL            ;посылаем в регистр маски прерываний
.
.
MOV  AL,0
OUT  21H,AL            ;очищаем IMR в конце программы
```

1.2.3. Написание собственного прерывания

Существует несколько причин для написания собственного прерывания. Во-первых, большинство из готовых прерываний, обеспечиваемых операционной системой, не что иное, как обычные процедуры, доступные для всех программ, и вы можете пожелать добавить свое в эту библиотеку. Например, многие ваши программы могут использовать процедуру, выводящую строки на экран вертикально. Вместо того чтобы включать ее в каждую программу в качестве процедуры, вы можете установить ее как прерывание, написав программу, которая останется резидентной в памяти после завершения [1.3.4]. Тогда вы можете использовать INT 80H вместо WRITE_VERTICALLY (имейте в виду, что вызов прерывания несколько медленней, чем вызов процедуры).

Во-вторых, причиной написания прерывания может быть использование какого-либо отдельного аппаратного прерывания. Это прерывание автоматически вызывается при возникновении определенных условий. В некоторых случаях BIOS инициализирует вектор этого прерывания так, что он указывает на процедуру, которая вообще ничего не делает (она содержит один оператор IRET). Вы можете написать свою процедуру и изменить вектор прерываний, чтобы он указывал на нее. Тогда при возникно-

вении аппаратного прерывания будет выполняться ваша процедура. Одна из таких процедур - это прерывание времени суток [2.1.0], которое автоматически вызывается 18,2 раза в секунду. Обычно это прерывание только обновляет показание часов, но вы можете добавить к нему любой код. Если ваш код проверяет показания часов и вступает в игру в определенные моменты времени, то возможны операции в реальном времени. Другие возможности - это написание процедур обработки «Ctrl-Break» [3.2.8], «PrtSc» [3.2.9] и возникновения ошибочных ситуаций [7.2.5]. Прерывания принтера [6.3.1] и коммуникационные [7.1.8] позволяют компьютеру быстро переключаться между операциями ввода/вывода и другой обработкой.

Наконец, вы можете захотеть написать прерывание, которое полностью заменит одну из процедур операционной системы, приспособленное к вашим программным нуждам. В [1.2.4] показано, как написать прерывание внутри прерывания, позволяющее модифицировать существующие процедуры.

Средний уровень

Функция 25H прерывания 21H устанавливает вектор прерывания на указанный адрес. Адреса имеют размер в два слова. Старшее слово содержит значение сегмента (CS), младшее содержит смещение (IP). Чтобы установить вектор, указывающий на одну из ваших процедур, нужно поместить сегмент процедуры в DS, а смещение в DX (следуя приведенному ниже примеру). Затем поместите номер прерывания в AL и вызовите функцию. Любая процедура прерывания должна завершаться не обычной инструкцией RET, а IRET. (IRET выталкивает из стека три слова, включая регистр флагов, в то время как RET помещает на стек только два. Если вы попытаетесь тестировать такую процедуру как обычную процедуру, но кончающуюся IRET, то вы исчерпаете стек.) Отметим, что функция 25H автоматически запрещает аппаратные прерывания в процессе изменения вектора, поэтому не существует опасности, что посреди дороги произойдет аппаратное прерывание, использующее данный вектор.

```
;---установка прерывания
        PUSH DS             ;сохраняем DS
        MOV  DX,OFFSET ROUT     ;смещение для процедуры в DX
        MOV  AX,SEG ROUT    ;сегмент процедуры
        MOV  DS,AX          ;помещаем в DS
        MOV  AH,25H         ;функция установки вектора
```

```
        MOV  AL,60H        ;номер вектора
        INT  21H           ;меняем прерывание
        POP  DS            ;восстанавливаем DS
;---процедура прерывания
ROUTINE PROC FAR
        PUSH AX            ;сохраняем все изменяемые регистры
        .
        .
        POP  AX            ;восстанавливаем регистры
        MOV  AL,20H        ;эти две строки надо использовать
        OUT  20H,AL        ;только для аппаратных прерываний
        IRET
ROUTINE ENDP
```

В конце кода каждого из ваших аппаратных прерываний вы должны включить следующие 2 строчки кода:

```
MOV  AL,20H
OUT  20H,AL
```

Это просто совпадение, что числа (20H) одни и те же в обеих строках. Если аппаратное прерывание не заканчивается этими строками, то микросхема 8259 не очистит информацию регистра обслуживания, с тем чтобы была разрешена обработка прерываний с более низкими уровнями, чем только что обработанное. Отсутствие этих строк легко может привести к сбою программы, так как прерывания от клавиатуры скорее всего будут замороженными и даже Ctrl-Alt-Del окажется бесполезным. Отметим, что эта добавка не нужна для векторов прерываний, являющихся расширениями существующих прерываний, таких, как прерывание 1CH, которое добавляет код к прерыванию времени суток [2.1.7].

Когда программа завершается, должны быть восстановлены оригинальные векторы прерываний. В противном случае последующая программа может вызвать данное прерывание и передать управление на то место в памяти, в котором вашей процедуры уже нет. Функция 35 прерывания 21H возвращает текущее значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX. Перед установкой своего прерывания получите текущее значение вектора, используя эту функцию, сохраните эти значения и затем восстановите их с помощью функции 25H (как выше) перед завершением своей программы. Например:

```
;---в сегменте данных:
KEEP_CS DW 0              ;хранит сегмент заменяемого прерывания
KEEP_IP  DW 0             ;хранит смещение прерывания
;---в начале программы
        MOV  AH,25H       ;функция получения вектора
        MOV  AL,1CH       ;номер вектора
        INT   21H         ;теперь сегмент в ES, смещение в BX
        MOV  KEEP_IP,BX ;запоминаем смещение
        MOV  KEEP_CS,ES ;запоминаем сегмент
;---в конце программы
        CLI
        PUSH DS           ;DS будет разрушен
        MOV  DX,KEEP_IP ;подготовка к восстановлению
        MOV  AX,KEEP_CS
        MOV  DS,AX        ;подготовка к восстановлению
        MOV  AH,25H       ;функция установки вектора
        MOV  AL,1CH       ;номер вектора
        INT   21H         ;восстанавливаем вектор
        POP   DS          ;восстанавливаем DS
        STI
```

Существуют "ловушки", которых следует избегать при написании прерывания. Если новая процедура прерывания должна иметь доступ к данным, то необходимо позаботиться, чтобы DS был правильно установлен (обычно прерывание может использовать стек вызывающей программы). Другая неприятность может заключаться в том, что при завершении программы по «Ctrl-Break» вектор прерывания не будет восстановлен, если только вы не предусмотрите, чтобы программа реакции на «Ctrl-Break» выполняла эту процедуру [3.2.8].

Низкий уровень

Описанные выше функции MS-DOS просто получают или изменяют пару слов в младших ячейках памяти. Смещение вектора может быть вычислено простым умножением номера вектора на 4. Например, чтобы получить адрес прерывания 16H в ES:BX, нужно:

```
;---получение адреса прерывания 16H
SUB   AX,AX               ;устанавливаем ES на начало памяти
```

```
MOV  ES,AX
MOV  DI,16H          ;номер прерывания в DI
SHL  DI,1            ;умножаем на 2
SHL  DI,1            ;умножаем на 2
MOV  BX,ES:[DI]      ;берем младший байт в BX
MOV  AX,ES:[DI] + 2  ;берем старший байт в ES
MOV  ES,AX
```

Не рекомендуется прямо устанавливать вектор прерываний, обходя функцию DOS. В частности, в многозадачной среде операционная система может поддерживать несколько таблиц векторов прерываний, и реальный физический адрес таблицы может быть известен только DOS.

1.2.4. Дополнение к существующему прерыванию

Хотя и не часто, но бывает полезно добавить код к существующему прерыванию. В качестве примера рассмотрим программы, которые преобразуют одно нажатие клавиши в длинные определяемые пользователем символьные строки (макроопределения клавиатуры). Эти программы используют тот факт, что весь ввод с клавиатуры поступает через функцию 0 прерывания 16H BIOS [3.1.3]. Все прерывания ввода с клавиатуры DOS вызывают прерывание BIOS для получения символа из буфера клавиатуры. Поэтому необходимо модифицировать лишь прерывание 16H таким образом, чтобы оно служило шлагбаумом для макроопределений, после чего любая программа будет получать макроопределения независимо от того, какое прерывание ввода с клавиатуры она использует.

Конечно, модифицировать прерывания BIOS и DOS непросто, поскольку BIOS расположена в ПЗУ, а DOS поступает без листинга и они ограничены размерами отведенной для них памяти. Но вы можете написать процедуру, которая предшествует и/или следует за соответствующим прерыванием, и эта процедура может вызываться при вызове прерывания DOS или BIOS. Например, в случае прерывания 16H вам нужно написать процедуру и указать на нее вектором прерывания для 16H. Оригинальное значение вектора 16H тем временем переносится в какой-либо неиспользуемый вектор, скажем 60H. Новая процедура просто вызывает прерывание 60H, чтобы использовать оригинальное прерывание 16H; поэтому когда программа вызывает прерывание 16H, управление передается вашей процедуре; она затем вызывает

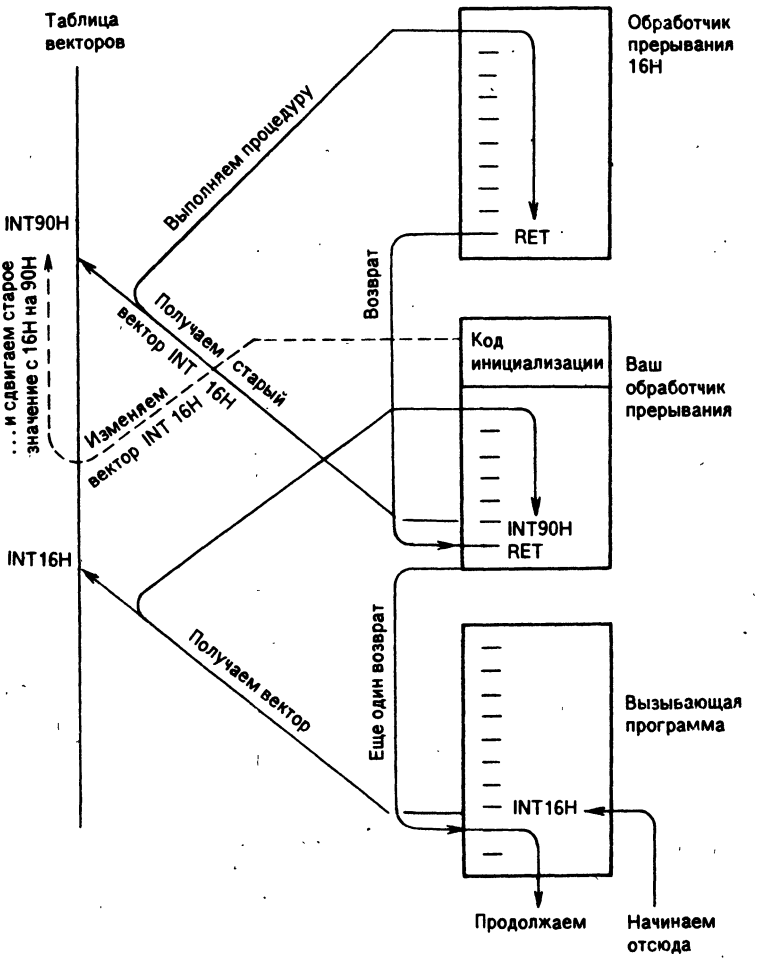


Рис. 1.3. Дополнение к существующему прерыванию

оригинальное прерывание 16H, которое по завершении снова возвращает управление вашей процедуре, а из нее вы возвращаетесь в то место программы, откуда поступил вызов прерывания 16H. После того как это сделано, в новой процедуре может содержаться любой код как до, так и после вызова

прерывания 60H. На рис. 1.3 показана диаграмма этой процедуры. Вот краткая сводка необходимых действий:

1. Создать новую процедуру, вызывающую прерывание 60H.
2. Перенести вектор прерывания для 16H в 60H.
3. Изменить вектор 16H, чтобы он указывал на новую процедуру.
4. Завершить программу, оставляя ее резидентной [1.3.4].

1.3. Управление программами

Большинство программ загружаются в память, запускаются, а затем удаляются операционной системой при завершении. Языки высокого уровня обычно не имеют альтернативы. Но для программистов на ассемблере существует другая возможность, и данный раздел демонстрирует ее. Некоторые программы действуют как драйверы устройств или драйверы прерываний и они должны быть сохранены в памяти ("резидентными") даже после их завершения (векторы прерываний обеспечивают механизм, посредством которого последующие программы могут обращаться к резидентным процедурам). Иногда программе необходимо запустить другую программу. DOS позволяет программе загрузить в память вторую копию COMMAND.COM, которая может быть использована как средство интерфейса с пользователем или выполнения команд типа COPY или DIR.

Программы могут быть в двух форматах: .EXE или .COM. Программы первого типа могут быть больше 64K, но они требуют некоторой обработки перед тем, как DOS загрузит их в память. С другой стороны, COM-программы существуют прямо в том формате, который нужен для загрузки в память. COM-программы особенно полезны для коротких утилит. В обоих случаях код, составляющий программу, предваряется в памяти префиксом программного сегмента (PSP). Это область размером 100H байт, которая содержит информацию, необходимую DOS для работы программы; PSP также обеспечивает место для файловых операций ввода/вывода [5.3.5]. При загрузке EXE-файла и DS, и ES указывают на PSP. Для COM-файлов CS также сначала указывает на PSP. Отметим, что MS-DOS 3.0 имеет функцию, которая возвращает номер сегмента PSP. Это функция 62H прерывания 21H; ей ничего не надо подавать на входе, а в BX возвращается номер параграфа.

Одна из причин, по которой интересно положение PSP, состоит в том, что его первое слово содержит номер прерывания DOS, которое будет приводить к завершению программы. Когда выполняется последний оператор RET программы, значения на вершине стека указывают счетчику команд (регистр IP) на начало PSP, таким образом, код завершения выполняется как следующая инструкция программы. Дальнейшее обсуждение см. в параграфах [1.3.4] и [1.3.6].

Для справки приводим значение полей PSP:

Смещение	Размер поля	Значение
0H	DW	номер функции DOS завершения программы
2H	DW	размер памяти в параграфах
4H	DW	резерв
6H	DD	длинный вызов функции диспатчера DOS
AH	DD	адрес завершения (IP,CS)
EH	DD	адрес выхода по Ctrl-Break (IP,CS)
12H	DD	адрес выхода по критической ошибке
16H	22 байта	резерв
2CH	DW	номер параграфа строки среды
2EH	46 байт	резерв
5CH	16 байт	область параметров 1 (формат FCB)
6CH	20 байт	область параметров 2 (формат FCB)
80H	128 байт	область DTA по умолчанию/получает командную строку программы

1.3.1. Манипуляции с памятью

Когда MS-DOS загружает программу, она помещается в младшую область памяти, сразу же за COMMAND.COM и установленными драйверами устройств или другими утилитами, которые резидентны в памяти. В этот момент вся память за программой отведена этой программе. Если программе нужна память для создания области данных, то она может приближенно вычислить, где в памяти кончается ее код, и затем поместить требуемую область данных в любое место за концом кода. Для определения адреса конца программы поместите в конце программы псевдосегмент типа

```
ZSEG SEGMENT
;
ZSEG ENDS
```

В ассемблере IBM PC ZSEG будет последним сегментом, так как сегменты располагаются в алфавитном порядке. В случае с другими ассемблерами нужно действительно поместить эти строки в конце программы. В самой программе достаточно поставить оператор MOV AX,ZSEG, и AX будет указывать на первый свободный сегмент памяти за программой.

Такой подход неприменим, если программа предполагает наличие памяти, которой на самом деле нет. Он не будет также работать в многопользовательской среде, когда несколько программ могут делить между собой одну и ту же область адресов. Для решения этой проблемы MS-DOS имеет возможность отслеживать 640K системной памяти и отводить по требованию программы блоки памяти любого размера. Блок памяти - это просто непрерывная область памяти, его максимальный размер определяется размером доступной памяти, в частности он может быть больше одного сегмента (64K). Если затребован слишком большой блок, то DOS выдает сообщение об ошибке. Любая возможность перекрытия блоков исключена. Кроме того, MS-DOS может освобождать, урезать или расширять существующие блоки. Хотя программа не обязана использовать эти средства, удобно и предусмотрительно делать это. Некоторые функции DOS требуют, чтобы были использованы средства управления памятью DOS, например завершение резидентной программы [1.3.4] или вызов другой программы из данной [1.3.2].

Прежде чем отвести память, существующий блок (вся память от начала программы до конца) должен быть обрезан до размера программы. Затем, при создании блока, DOS создает 16-байтовый управляющий блок памяти, который расположен непосредственно перед блоком памяти. Первые 5 байт этого блока имеют следующее значение:

байт	0	ASCII 90, если последний блок в цепочке, иначе - ASCII 77
	1-2	0, если блок освобожден
	3-4	размер блока в 16-байтовых параграфах

DOS обращается к блокам по цепочке. Адрес первого блока хранится во внутренней переменной. Значение этой переменной позволяет DOS определить положение первого отведенного блока, а из информации, содержащейся в нем, может быть найден

следующий блок и т.д., как показано на рис. 1.4. Как только вы начали использовать систему распределения памяти DOS, вы обязаны придерживаться ее. Если программа изменит содержимое управляющего блока, цепочка будет разорвана и DOS начнет выдавать сообщения об ошибке.

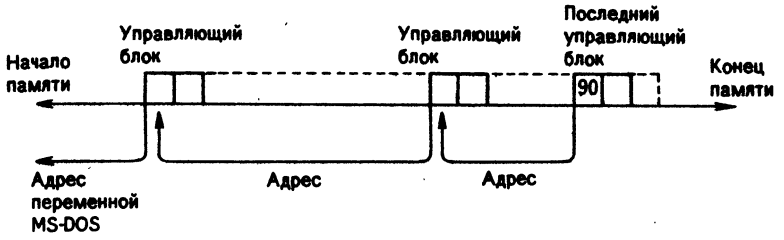


Рис. 1.4. Структура управляющего блока памяти

MS-DOS обеспечивает три функции распределения памяти, номера от 48H до 4AH прерывания 21H. Функция 48H отводит блок памяти, а 49H освобождает блок памяти. Третья функция (SETBLOCK) меняет размер памяти, отведенной для программы; эта функция должна быть использована перед двумя остальными. После ее выполнения можно спокойно отводить и освобождать блоки памяти. Программа должна освободить все отведенные ею блоки перед завершением. Иначе эта память станет недоступной при последующей работе.

Средний уровень

Все три функции распределения памяти прерывания 21H используют 16-битовый адрес начала блока памяти, с которым они оперируют. Этот адрес соответствует сегменту, с которого начинается блок (блок всегда начинается со смещения 0 данного сегмента). Таким образом, реальный адрес ячейки начала блока равен этому адресу, умноженному на 16. Для всех трех функций BX содержит число 16-байтовых разделов памяти (параграфов), которые будут отводиться или освобождаться. Если функция не может быть выполнена, то устанавливается флаг переноса, а в AX возвращается код ошибки, объясняющий причину. Возможны три кода ошибки:

7 разрушен управляющий блок памяти
8 недостаточно памяти для выполнения функции
9 неверный адрес блока памяти

Функция отведения блока использует коды 7 и 8, а освобождения - 7 и 9, в то время как функция изменения блока использует все три кода. В следующем примере сначала отводится блок размером 1024 байта. При этом BX содержит требуемое число 16-байтовых параграфов, а при завершении стартовый адрес блока равен AX:0 (т.е. смещение 0 в сегменте со значением, содержащимся в AX). Вторая часть примера освобождает этот же блок, как и требуется при завершении программы. В данном случае значение, полученное в AX, помещается в ES. DOS следит за размером блока и знает, какое количество параграфов надо освободить.

```
;---отведение блока размером 1024 байта
MOV  AH,48H          ;номер функции
MOV  BX,64           ;требуем 64 параграфа
INT  21H             ;пытаемся отвести блок
JC   ERROR           ;обрабатываем ошибку в случае неудачи
MOV  BLOCK_SEG,AX    ;иначе, сохраняем адрес блока
;---освобождаем тот же блок
MOV  AX,BLOCK_SEG    ;получаем стартовый адрес блока
MOV  ES,AX           ;помещаем его в ES
MOV  AH,49H          ;номер требуемой функции
INT  21H             ;освобождаем блок памяти
```

Наконец, приведем пример использования функции 4AH. ES содержит значение сегмента PSP, т.е. самого первого байта памяти, с которого загружена программа. Это значение присваивается ES при старте задачи. Для использования SETBLOCK надо либо вызывать эту функцию в самом начале программы (прежде чем ES будет изменен), либо сохранить его начальное значение.

BX содержит требуемый размер блока в 16-байтовых параграфах. Для определения этого размера поместите добавочный "исскуственный" сегмент в конец программы. В макроассемблере IBM PC сегменты располагаются в алфавитном порядке, поэтому вы можете поместить его в любое место программы при условии, что его имя - это что-то вроде "ZSEG". (В других ассемблерах действительно помещайте фиктивный сегмент в конец программы.)

Программа может прочитать позицию этого сегмента и, сравнивая ее со стартовым сегментом, получить количество памяти, требуемое самой программе. В момент загрузки программы и ES, и DS содержат номер параграфа самого начала программы в префиксе программного сегмента; для COM-файлов CS также указывает на эту позицию, но для EXE-файлов это не так.

```
;---освобождение памяти (ES имеет значение при старте)
MOV  BX,ZSEG          ;получаем # параграфа конца программы + 1
MOV  AX,ES            ;получаем # параграфа начала программы
SUB  BX,AX            ;вычисляем размер программы в параграфах
MOV  AH,4AH           ;номер функции
INT  21H              ;освобождаем память
JC   MEMORY_ERROR     ;проверяем на ошибку
;---пустой сегмент
ZSEG SEGMENT
ZSEG ENDS
```

1.3.2. Запуск одной программы из другой

MS-DOS обеспечивает функцию EXEC (номер 4BH прерывания 21H), реализующую вызов одной программы из другой. Первая программа называется "родителем", а загружаемая и запускаемая - "потомком".

Высокий уровень

В Бейсике версии 3.0 введена команда SHELL. Со значительными ограничениями она позволяет бейсиковской программе загрузить и выполнить другую программу. Формат этой команды - SHELL ком_строка. Командная строка может быть просто именем программы или она может содержать, кроме имени, параметры, которые обычно следуют за именем программы в командной строке. Если ком_строка не указана, то загружается копия COMMAND.COM и появляется запрос операционной системы. В этот момент можно выполнить любую команду MS-DOS, а по завершении вернуть управление бейсиковской программе, введя команду EXIT.

Имеется ряд ограничений при использовании SHELL. Если загружаемая программа меняет режим работы дисплея, то он не будет автоматически восстановлен при возврате. Перед загрузкой программы все файлы должны быть закрыты. Программа, которая остается резидентной после завершения, не может быть запущена

командой SHELL. Обсуждение ряда других проблем содержится в руководстве по Бейсику.

Средний уровень

Применение функции 4BH более сложно, чем применение остальных, оно требует четырех подготовительных шагов.

1. Подготовить в памяти место, доступное программе.
2. Создать блок параметров.
3. Построить строку, содержащую накопитель, путь и имя программы.
4. Сохранить значения регистров SS и SP в переменных.

Поскольку при загрузке программы MS-DOS выделяет ей всю доступную память, необходимо освободить место в памяти. Если не освободить часть памяти, то не будет места для загрузки второй программы. В [1.3.1] объяснено, как это сделать с помощью функции SETBLOCK. После того как память освобождена, вы должны просто поместить в BX требуемое число 16-байтовых параграфов, заслать 4AH в AH и выполнить прерывание 21H, делая доступным программе именно то количество параграфов, которое ей необходимо.

Блок параметров, на который должны указывать ES:BX, это 14-байтовый блок памяти, в который вы должны поместить следующую информацию:

```
DW  сегментный адрес строки среды
DD  сегмент и смещение командной строки
DD  сегмент и смещение первого FCB
DD  сегмент и смещение второго FCB
```

Строка среды - это строка, состоящая из одной или более спецификаций, которым следует MS-DOS при выполнении программы. Элементы строки среды такие же, как и те, что можно обнаружить в дисковом файле CONFIG.SYS. Например, в строку может быть помещено VERIFY = ON. Начните строку с первого элемента, завершив его символом ASCII 0, потом запишите следующий и т.д. За последним элементом должны следовать два символа ASCII 0. Строка должна начинаться на границе параграфа (т.е. ее адрес по модулю 16 должен быть равен нулю). Это вызвано тем, что соответствующий вход в блоке параметров, указывающий на строку, содержит только 2-байтовое сегментное значение. Все это не нужно, если новая программа может работать с той же строкой среды, что и программа "родитель". В этом

случае надо просто поместить два символа ASCII 0 в первые 2 байта блока параметров.

Следующие 4 байта блока параметров указывают на командную строку для загружаемой программы. "Командная строка" - это символьная строка, определяющая способ работы программы. При загрузке программы из DOS она может иметь вид вроде EDITOR A:CHAPTER1\NOTES.MS. При этом вызывается редактор и ему передается имя файла в подкаталоге накопителя A для немедленного открытия. Когда вы подготавливаете командную строку для EXEC, надо включать только последнюю часть информации, но не имя загружаемой программы. Перед командной строкой должен стоять байт, содержащий длину этой строки, и она должна завершаться символом «CR» (ASCII 13).

Последние 8 байт блока параметров указывают на управляющие блоки файлов (FCB). FCB содержит информацию об одном или двух файлах, указанных в командной строке. Если открываемых файлов нет, то надо заполнить все 8 байт символом ASCII 0. В [5.3.5] объяснено, как работает FCB. Начиная с версии MS-DOS 2.0, использование FCB необязательно и вы можете не включать информацию FCB, вместо этого используя новую конвенцию дескриптора файлов (file handle), в которой доступ к файлу предоставляется по кодовому номеру, а не через FCB (см.также [5.3.5]).

Наконец, вы должны построить строку с указанием накопителя, пути и имени файла. Эта строка именует загружаемую программу. DS:DX указывает на нее при выполнении EXEC. Эта строка - стандартная строка ASCIIZ, т.е. стандартная спецификация файла, завершаемая кодом ASCII 0. Например, это может быть B:\NEWDATA\FILER.EXE«NUL», где символом «NUL» обозначен код ASCII 0.

После того как вся указанная информация подготовлена, остается последняя задача. Поскольку все регистры будут изменены вызываемой задачей, надо сохранить сегмент стека и указатель стека, с тем чтобы они могли быть восстановлены, когда управление будет возвращено вызвавшей задаче. Для их сохранения создайте переменные. Поскольку значение регистра DS также будет изменено, эти переменные не могут быть найдены, до тех пор пока DS не будет восстановлен операторами MOV AX,DSEG и MOV DS,AX. После того как SS и SP запомнены, поместите 0 в AL для выбора операции "загрузка и запуск" (EXEC используется также для оверлеев [1.3.5]). Затем поместите 4AH в AH и вызовите прерывание 21H. В этот момент запущены две программы, причем программа "родитель" находится в остановленном состоянии. MS-DOS предоставляет возможность программе "потомок" передать "родителю" код возврата, таким

образом, могут быть переданы ошибки и статус. В [7.2.5] объяснено, как это сделать. Что касается самой функции запуска, то при возникновении ошибки устанавливается флаг переноса, а регистр AX в этом случае будет возвращать 1 для неправильного номера функции, 2 - если файл не найден, 5 - при дисковой ошибке, 8 - при нехватке памяти, 10 - если неправильна строка среды и 11 - если неверен формат.

Приводимый пример - простейший из возможных, но часто большего для процедуры EXEC и не требуется. Здесь оставлен нулевым блок параметров и не создана строка среды. Это означает, что загружаемой программе не передается командная строка и что среда будет такой же, как и для вызывающей программы. Вы должны только изменить распределение памяти, создать имя и (пустой) блок параметров и сохранить значения SS и SP.

```
;---в сегменте данных
FILENAME    DB 'A:TRIAL.EXE',0 ;загружаем TRIAL.EXE
PARAMETERS DW 7DUP(0)          ;нулевой блок параметров
KEEP_SS     DW 0               ;переменная для SS
KEEP_SP     DW 0               ;переменная для SP
;---перераспределение памяти
MOV  BX,ZSEG                   ;получить # параграфа конца
MOV  AX,ES                     ;получить # параграфа начала
SUB  BX,AX                     ;вычислить размер программы
MOV  AH,4AH                    ;номер функции
INT  21H                       ;перераспределение
;---указываем на блок параметров
MOV  AX,SEG PARAMETERS         ;в ES - сегмент
MOV  ES,AX
MOV  BX,OFFSET PARAMETERS      ;в BX - смещение
;---сохранить копии SS и SP
MOV  KEEP_SS,SS                ;сохраняем SS
MOV  KEEP_SP,SP                ;сохраняем SP
;---указываем на строку имени файла
MOV  DX,OFFSET FILENAME        ;смещение - в DX
MOV  AX,SEG FILENAME           ;сегмент - в DS
MOV  DS,AX
;---загрузка программы
MOV  AH,4BH                    ;функция EXEC
MOV  AL,0                      ;выбираем "загрузку и запуск"
```

```
INT   21H                          ;запускаем задачу
;---впоследствии, восстанавливаем регистры
MOV   AX,DSEG                      ;восстанавливаем DS
MOV   DS,AX
MOV   SS,KEEP_SS                   ;восстанавливаем SS
MOV   SP,KEEP_SP                   ;восстанавливаем SP
;---в конце программы создаем фиктивный сегмент
ZSEG SEGMENT                       ;см. [1.3.1]
ZSEG ENDS
```

1.3.3. Использование команд операционной системы из программы

Программа может иметь в своем распоряжении полный набор команд интерфейса с пользователем DOS, таких, как DIR или CHKDSK. Когда эти команды используются из программы, загружается и запускается вторая копия COMMAND.COM. Хотя такой подход при программировании может сэкономить много усилий, его успешная реализация требует достаточного количества памяти для второй копии и ваша программа может попасть в "ловушку", если памяти недостаточно.

Высокий уровень

Бейсик 3.0 позволяет загрузить вторую копию COMMAND.COM с помощью оператора SHELL. SHELL обсуждается в [1.3.2]. COMMAND.COM загружается, когда не указано имя файла, поэтому, вводя просто SHELL, вы получаете запрос MS-DOS. В этот момент можно использовать любую из утилит DOS, включая командные файлы. Для возврата в вызвавшую программу надо ввести EXIT.

Средний уровень

В этом случае к примеру, приведенному в [1.3.2], нужно добавить командную строку. Обычно она начинается с байта длины строки, затем следует сама командная строка и, наконец, код ASCII 13. При передаче команды COMMAND.COM вы должны указать /C перед строкой (см. раздел "Вызов вторичного командного процессора" руководства по MS-DOS). Вы должны также указать накопитель, на котором находится COMMAND.COM, поместив имя накопителя в начале командной

строки. Чтобы вывести каталог накопителя A:, а COMMAND.COM при этом находится на накопителе B:, нужна строка:

```
COMMAND_LINE DB 12,'B: /C DIR A:',13
```

Следующий кусочек кода устанавливает адрес командной строки в блок параметров, используемый в примере [1.3.2]:

```
LEA   BX,PARAMETERS              ;получение адреса блока пар-ров
MOV   AX,OFFSET COMMAND_LINE     ;получение смещения командной строки
MOV   [BX] + 2,AX                ;пересылка в 1-е 2 байта блока
MOV   AX,SEG COMMAND_LINE        ;получение сегмента командной строки
MOV   [BX] + 4,AX                ;пересылка во 2-е 2 байта блока
```

1.3.4. Сохранение программы в памяти после завершения

Программы, оставленные резидентными в памяти, могут служить в качестве утилит для других программ. Обычно такие программы вызываются через неиспользуемый вектор прерывания. MS-DOS рассматривает такие программы как часть операционной системы, защищая их от наложения других программ, которые будут загружены впоследствии. Резидентные программы обычно пишутся в форме COM, что обсуждается в [1.3.6]. Программы, написанные в форме EXE, оставить резидентными в памяти немного труднее.

Завершение программы прерыванием 27H оставляет ее резидентной в памяти. CS должен указывать на начало PSP для того, чтобы эта функция работала правильно. В программах COM CS сразу устанавливается соответствующим образом, поэтому надо просто завершить программу прерыванием 27H. В программах EXE CS первоначально указывает на первый байт, следующий за PSP (т.е. 100H). При нормальном завершении EXE-программы последняя инструкция RET выталкивает из стека первые положенные туда значения: PUSH DX / MOV AX,0 / PUSH AX. Поскольку DS первоначально указывает на начало PSP, при получении этих значений из стека счетчик команд указывает на смещение 0 в PSP, где при инициализации записывается инструкция INT 20H. Поэтому INT 20H выполняется, а это стандартная функция для завершения программы и передачи управления в DOS. На рис. 1.5 показан этот процесс. Чтобы заставить прерывание 27H работать в EXE-программе, надо поместить 27H во второй байт PSP (первый содержит машинный код инструкции INT), а затем завершить программу обычным

RET. Для обоих типов файлов, прежде чем выполнить прерывание 27H, DX должен содержать смещение конца программы, отсчитываемое от начала PSP.

Рис. 1.5. Завершение EXE-программы

Средний уровень

Вектор прерывания устанавливается с помощью функции 25H прерывания 21H, как показано в [1.2.3] (здесь используется вектор 70H). Позаботьтесь, чтобы процедура оканчивалась IRET. Кроме самой процедуры, устанавливаемая программа не должна делать ничего, помимо инициализации вектора прерывания, присвоения DX значения смещения конца процедуры и завершения. Для COM-файлов просто поместите оператор INT 27H в конец программы. Для EXE-файлов поместите этот оператор в первое слово PSP и завершите программу обычным оператором RET. Для того чтобы выполнить процедуру, впоследствии загруженная программа должна вызвать INT 70H.

Ниже приведены примеры для обоих типов файлов (COM и EXE). В них установлена метка FINISH для отметки конца процедуры прерывания (напоминаем, что знак $ дает значение счетчика команд в этой точке). Для COM-файлов FINISH дает смещение от начала PSP, как и требуется для прерывания 27H.

Для EXE-файлов смещение отсчитывается от первого байта, следующего за PSP, поэтому к нему необходимо прибавить 100H, чтобы пересчитать на начало PSP. Заметим, что, поместив процедуру в начало программы, мы можем исключить установочную часть кода из резидентной порции. Другой возможный фокус состоит в использовании инструкции MOVSB для пересылки кода процедуры вниз в неиспользуемую часть PSP, начиная со смещения 60H, что освобождает 160 байт памяти.

Файл COM:

```
;---здесь процедура прерывания
BEGIN:   JMP    SHORT SET_UP        ;переход на установку
ROUTINE PROC  FAR
         PUSH   DS                  ;сохранение регистров
.
(процедура)
.
         POP    DS                  ;восстановление регистров
         IRET                       ;возврат из прерывания
FINISH   EQU    $                   ;отметка конца процедуры
ROUTINE ENDP
;---установка вектора прерывания
SET_UP:  MOV    DX,OFFSET ROUTINE   ;смещение процедуры в DX
         MOV    AL,70H              ;номер вектора прерывания
         MOV    AH,25H              ;функция установки вектора
         INT    21H                 ;устанавливаем вектор
;---завершение программы, оставляя резидентной
         LEA    DX,FINISH           ;определяем требуемое смещение
         INT    27H                 ;завершение
```

Файл EXE:

```
;---здесь резидентная процедура
         JMP    SHORT SET_UP        ;переход на установку
ROUTINE  PROC   FAR
         PUSH DS                    ;сохранение регистров
         .
         (процедура)
         .
         POP    DS                  ;восстановление регистров
         IRET                       ;возврат из прерывания
FINISH:  EQU    $                   ;отметка конца процедуры
ROUTINE  ENDP
;---установка вектора прерывания
```

```
SET_UP:  MOV   DX,OFFSET ROUTINE;смещение процедуры в DX
         MOV   AX,SEG ROUTINE    ;сегмент процедуры в DS
         MOV   DS,AX
         MOV   AL,70H            ;номер вектора прерывания
         MOV   AH,25H            ;функция установки вектора
         INT   21H               ;установка вектора
;---завершение программы
         MOV   DX,FINISH + 100H  ;вычисляем смещение конца
         MOV   BYTE PTR ES:1,27H ;посылаем 27H в PSP
         RET                     ;завершаем процедуру
```

Функция 31H прерывания 21H работает аналогично, за исключением того, что в DX должно содержаться число 16-байтовых параграфов, требуемых процедуре (вычисление размера процедуры от начала PSP, см. в примере из [1.3.1]). Преимуществом этой функции является то, что она передает родительской программе код выхода, дающий информацию о статусе процедуры. Родительская программа получает этот код с помощью функции 4DH прерывания 21H. Коды выхода обсуждаются в [7.2.5].

1.3.5. Загрузка и запуск программных оверлеев

Оверлеи - это части программы, которые остаются на диске, в то время как тело программы резидентно в памяти. Когда требуется функция, выполняемая каким-либо оверлеем, он загружается в память и программа вызывает его как процедуру. Различные оверлеи могут загружаться в одно и то же место памяти, перекрывая предыдущий код. Например, программа ведения базы данных может загрузить процедуру сортировки, а затем перекрыть ее процедурой генерации отчетов. Эта техника применяется для экономии памяти. Но она хороша только для тех процедур, которые не используются постоянно, иначе частые обращения к диску приведут к тому, что программа будет выполняться слишком медленно.

Средний уровень

MS-DOS использует функцию EXEC для загрузки оверлеев. Эта функция, номер 4BH прерывания 21H, служит также для загрузки и запуска одной программы из другой, если поместить код 0 в AL [1.3.2]. Если в AL поместить код 3, то будет загружен оверлей. В этом случае не создается PSP, поэтому оверлей как

независимая программа не устанавливается. Такая процедура просто загружает оверлей, не передавая ему управления.

Существуют два способа обеспечить память для оверлея. Может быть использована либо область внутри тела программы, либо специально отведенная область памяти за пределами головной программы. В качестве позиции функции EXEC передается только сегментный адрес, по которому будет загружен оверлей. Когда оверлей загружается в тело головной программы, программа сама должна вычислить номер параграфа, куда будет загружаться оверлей. С другой стороны, при загрузке в специально отведенную память MS-DOS обеспечивает программу номером параграфа.

В приведенном ниже примере используется загрузка в отведенную память. Поскольку DOS отводит программе всю доступную память, сначала необходимо освободить память с помощью функции 4AH. Функция 48H отводит достаточно большой блок памяти, чтобы он мог принять самый большой из оверлеев. Эта функция возвращает значение сегмента блока в AX, и этот номер параграфа определяет, куда будет загружен оверлей, а также по какому адресу оверлей будет вызываться головной программой. Эти функции детально обсуждаются в [1.3.1].

Кроме кода 3, засылаемого в AL, вы должны установить для этой функции еще два параметра. DS:DX должны указывать на строку, дающую путь к файлу оверлея, которая завершается байтом ASCII 0. Необходимо указывать полное имя файла, включая расширение .COM или .EXE, поскольку DOS в данном случае не считает, что он ищет программный файл.

Наконец, ES:BX должны указывать на 4-байтовый блок параметров, который содержит 2-байтовый номер параграфа, куда будет загружаться оверлей и 2-байтовый фактор привязки адресов в оверлее (привязка объясняется в [1.3.6]). В качестве номера параграфа надо использовать число, возвращаемое в AX, для номера параграфа отведенного блока памяти. Фактор привязки дает смещение, по которому могут быть вычислены адреса требующих привязки параметров в оверлее. Используйте номер параграфа, куда загружается оверлей. После того как он установлен, вызовите функцию и оверлей будет загружен. Изменяя путь к оверлейному файлу, можно вновь и вновь вызывать эту функцию, загружая все новые и новые оверлеи. Если при возврате установлен флаг переноса, то была ошибка и ее код будет возвращен в AX. Код равен 1, если указан неверный номер функции, 2 - если файл не найден, 5 - при дисковых ошибках и 8 - при отсутствии достаточной памяти.

После того как оверлей загружен в память, к нему можно получить доступ как к далекой (far) процедуре. В сегменте данных

должен быть установлен двухсловный указатель, определяющий этот вызов. Сегментная часть указателя равна текущему кодовому сегменту. Смещение оверлея должно быть вычислено нахождением разницы между сегментами кода и оверлея и умножением результата на 16 (при переводе величины из параграфов в байты). В приведенном примере две переменные OVERLAY_OFFSET и CODE_SEG помещены одна за другой для правильной установки указателя. Однажды загруженный оверлей затем может вызываться инструкцией CALL DWORD PTR OVERLAY_OFFSET.

Оверлей может быть полной программой со своими сегментами данных и стека, хотя, как правило, используется стековый сегмент вызывающей программы. При вызове оверлея значение сегмента его собственного сегмента данных должно быть помещено в DS.

```
;---завершаем программу фиктивным сегментом (см. [1.3.1])
ZSEG            SEGMENT
ZSEG            ENDS
;---в сегменте данных
OVERLAY_SEG     DW ?
OVERLAY_OFFSET  DW ?                    ;смещение оверлея
CODE_SEG        DW ?                    ;сегмент оверлея - должен
PATH            DB 'A:OVERLAY.EXE'      ;следовать за смещением
0BLOCK          DD 0                    ;4-байтовый блок параметров
;---освобождаем память
MOV  CODE_SEG,CS            ;создаем копию CS
MOV  AX,ES                  ;копируем значение сегмента PSP
MOV  BX,ZSEG                ;адрес сегмента конца программы
SUB  BX,AX                  ;вычисляем разность
MOV  AH,4AH                 ;номер функции SETBLOCK
INT  21H                    ;освобождаем память
JC   SETBLK_ERR             ;флаг переноса говорит об ошибке
;---отводим память для оверлея
MOV  BX,100H                ;отводим для оверлея 1000H байт
MOV  AH,48H                 ;функция отведения памяти
INT  21H                    ;теперь AX:0 указывает на блок
JC   ALLOCATION_ERR         ;флаг переноса говорит об ошибке
MOV  OVERLAY_SEG,AX         ;запасаем адрес сегмента оверлея
;---вычисление смещения оверлея в кодовом сегменте
MOV  AX,CODE_SEG            ;вычитаем значение сегмента оверлея
MOV  BX,OVERLAY_SEG         ;из значения сегмента кода
SUB  BX,AX                  ;BX содержит число параграфов
```

```
MOV  CL,4                    ;сдвигаем это число на 4 бита влево
SHL  BX,CL                   ;чтобы получить величину в байтах
MOV  OVERLAY_OFFSET,BX       ;запоминаем смещение
;---загрузка первого оверлея
MOV  AX,SEG BLOCK            ;ES:BX указывает на блок параметров
MOV  ES,AX
MOV  BX,OFFSET BLOCK
MOV  AX,OVERLAY_SEG          ;помещаем адрес сегмента оверлея в
MOV  [BX],AX                 ;первое слово блока параметров
MOV  [BX] + 2,AX             ;сегмент оверлея - фактор привязки
LEA  DX,PATH                 ;DS:DX указывает на путь к файлу
MOV  AH,48H                  ;номер функции EXEC
MOV  AL,3                    ;код загрузки оверлея
INT  21H                     ;загружаем оверлей
JC   LOAD_ERROR              ;флаг переноса говорит об ошибке
;---теперь программа занимается своими делами
.
.
CALL DWORD PTR OVERLAY_OFFSET ;вызов оверлея
.                             ;нужно указывать DWORD PTR, так как
.                             ;оверлей - далекая процедура
;---посмотрите эту структуру, когда будете писать оверлей
DSEG     SEGMENT              ;как обычно, устанавливаем сегмент данных
.                             ;опускаем стековый сегмент (используется
.                             ;стек вызывающей программы)
DSEG     ENDS
CSEG     SEGMENT PARA PUBLIC 'CODE'
OVERLAY  PROC    FAR          ;всегда далекая процедура
         ASSUME CS:CSEG,DS:DSEG
         PUSH    DS           ;храним DS вызывающей программы
         MOV     AX,DSEG      ;устанавливаем DS оверлея
         MOV     DS,AX
.
.
         POP     DS           ;восстанавливаем DS при завершении
         RET
OVERLAY  ENDP
CSEG     ENDS
         END
```

1.3.6. Преобразование программ из типа .EXE в тип .COM

Программисты на ассемблере имеют возможность преобразовать свои программы из обычного формата EXE в формат COM. Файлы EXE имеют заголовок, содержащий информацию для привязки; DOS привязывает некоторые адреса программы при загрузке. С другой стороны, файлы COM существуют в таком виде, что привязки не требуется, они хранятся уже в том виде, в котором загружаемая программа должна быть в памяти машины. По этой причине файлы EXE по меньшей мере на 768 байт больше на диске, чем их COM-эквиваленты (хотя при загрузке в память они будут занимать одинаковое место). Файлы COM также быстрее загружаются, поскольку не требуется привязки. Других преимуществ у них нет, а некоторые программы слишком сложны и слишком велики, чтобы их можно было преобразовать в тип COM.

Привязка - это процесс установки адресов, связанных с сегментным регистром. Например, программа может указывать на начало области данных следующим кодом:

```
MOV  DX,OFFSET DATA_AREA
MOV  AX,SEG DATA_AREA
MOV  DS,AX
```

Смещение в DX связано с установкой сегментного регистра DS. Но какое значение должен принимать сам DS? Программа требует абсолютный адрес, но номер параграфа, в котором будет располагаться DATA_AREA, зависит от того, в какое место в памяти будет загружена программа, а это зависит от версии MS-DOS и от того, какие резидентные программы будут находиться в младших адресах памяти. По этой причине во время компоновки программы можно только установить некоторые сегментные значения через смещения относительно начала программы. Затем, когда DOS осуществляет привязку, значение начального адреса программы прибавляется к сегментным значениям, давая абсолютные адреса, требуемые в сегментном регистре. На рис. 1.6 показан процесс привязки.

Файлы COM не нуждаются в привязке, поскольку они хранятся в таком виде, что им не требуется фиксация сегмента. Все в программе хранится относительно начала кодового сегмента, включая все данные и стек. По этой причине программа не может превышать 65535 байт по длине, что соответствует максимальному смещению, которое существует в используемой схеме адресации (поскольку верхняя часть этого блока занята стеком, реальное

пространство, доступное для кода и данных, немного меньше, чем 65535 байт, хотя стековый сегмент при необходимости может быть вынесен за границу 64-килобайтового блока). В файлах COM все сегментные регистры указывают на начало PSP; сравните с файлами EXE, где DS и ES инициализируются аналогичным образом, но CS указывает на первый байт, следующий за PSP.

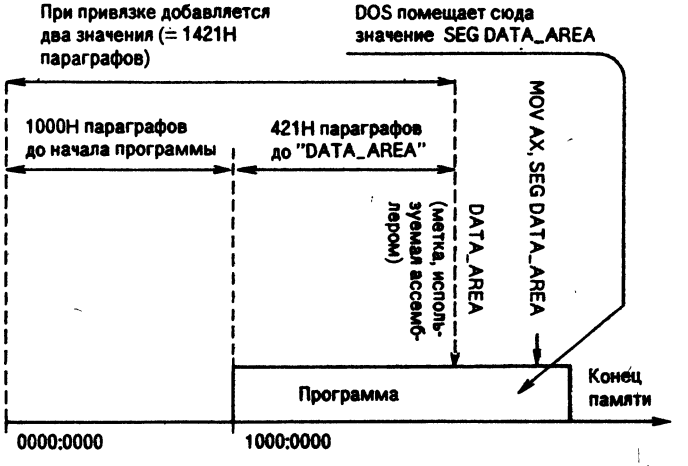


Рис. 1.6. Привязка значений памяти DATA_AREA

Для представления программы в виде файла COM требуется соблюдение следующих правил:

1. Не оформляйте программу в виде процедуры. Вместо этого поместите в самое начало метку, вроде START, и завершите программу оператором END START.
2. Поместите в начале программы оператор ORG 100H. Этот оператор указывает начало кода (т.е. устанавливает счетчик команд). Программы COM начинаются с 100H, что является первым байтом, следующим за PSP, поскольку CS указывает на начало PSP, которое расположено на 100H байт ниже. Для того чтобы начать выполнение с любого другого места, поместите по адресу 100H инструкцию JMP.
3. Оператор ASSUME должен устанавливать DS, ES и SS таким образом, чтобы они совпадали со значением для кодового сегмента, например ASSUME CS:CSEG, DS:CSEG, ES:CSEG, SS:CSEG.

4. Данные программы могут помещаться в любом месте программы до тех пор, пока они не перемешаны с кодом. Лучше начинать программы с области данных, поскольку макроассемблер может выдавать сообщения об ошибках при первом проходе, если имеются ссылки на идентификатор данных, который еще не обнаружен. Для перехода к началу кода используйте в качестве первой команды программы инструкцию JMP.

5. Нельзя применять фиксацию сегментов типа MOV AX,SEG NEW_DATA. Достаточно указания одного смещения метки. В частности, нужно опускать обычный код, используемый в начале программы для установки сегмента данных, MOV AX,DSEG / MOV DS,AX.

6. Стековый сегмент полностью опускается в начальном коде. Указатель стека инициализируется на вершину адресного пространства 64K, используемого программой (напоминаем, что стек растет вниз в памяти). В программах COM он должен быть сделан меньше, чем 64K, SS и SP могут быть изменены. Имейте в виду, что при компоновке программы компоновщик выдаст сообщение об ошибке, указывающее, что сегмент стека отсутствует. Игнорируйте его.

7. Завершите программу либо инструкцией RET, либо прерыванием 20H. Прерывание 20H - это стандартная функция для завершения программы и возврата управления в DOS. Даже когда программа завершается инструкцией RET, на самом деле используется прерывание 20H. Это происходит потому, что вершина стека первоначально содержит 0. При выполнении завершающей инструкции программы RET 0 выталкивается из стека, переназначая счетчик команд на начало PSP. Находящаяся в этой ячейке функция 20H выполняется как следующая инструкция программы, вызывая передачу управления в DOS. Все это означает, что вам не надо при старте программы помещать на стек DS и 0 (PUSH DS / MOV AX,0 / PUSH AX), как это требуется для EXE-файлов.

После того как программа сконструирована таким образом, ассемблируйте и компонуйте ее как обычно. Затем преобразуйте ее в форму COM с помощью утилиты EXE2BIN, имеющейся в MS-DOS. Если имя программы, построенной компоновщиком, MYPROG.EXE, то просто введите команду EXE2BIN MYPROG. В результате вы получите программный файл с именем MYPROG.BIN. Все что вам останется теперь сделать, - переименовать этот файл в MYPROG.COM. Вы можете также сразу использовать команду EXE2BIN MYPROG MYPROG.COM для получения файла с расширением COM.

Низкий уровень

В данном примере содержится законченная короткая программа, которая по установке переключателей определяет количество накопителей в машине и затем выводит сообщение на экран. Она может служить примером короткой утилиты, для которой формат COM идеален.

```
CSEG    SEGMENT
        ORG  100H
        ASSUME CS:CSEG, DS:CSEG, SS:CSEG
;---данные
START:  JMP  SHORT BEGIN ;переход к коду
MESSAGE1 DB 'The dip switches are set for $'
MESSAGE2 DB 'disk drive(s).$'
;---печать первой части сообщения
BEGIN:  MOV  AH,9              ;функция 9 прерывания 21H - вывод
        MOV  DX,OFFSET MESSAGE1 ;строки
        INT  21H               ;выводим строку
        PUSH AX                ;сохраняем номер функции на будущее
;---получаем установку переключателей из порта A микросхемы 8255
        IN   AL,61H            ;получаем байт из порта B
        OR   AL,10000000B      ;устанавливаем бит 7
        OUT  61H,AL            ;заменяем байт
        IN   AL,60H            ;получаем установку переключателей
        AND  AL,11000000B      ;выделяем старшие 2 бита
        MOV  CL,6              ;подготовка к сдвигу AL вправо
        SHR  AL,CL             ;сдвигаем 2 бита в начало
        ADD  AL,49             ;добавляем 1, чтобы считать с 1
                               ;и 48 для перевода в ASCII
        MOV  DL,AL             ;помещаем результат в DL
        MOV  AL,61H            ;должны восстановить порт B
        AND  AL,01111111B      ;сбрасываем бит 7
        OUT  61H,AL            ;возвращаем байт
;---печать числа накопителей
        MOV  AH,2              ;функция 2 прерывания 21H
        INT  21H               ;печатаем число из DL
;---печать второй половины сообщения
        POP  AX                ;берем номер функции со стека
        MOV  DX,OFFSET MESSAGE2
        INT  21H               ;выводим строку
        INT  20H               ;завершение программы
CSEG    ENDS
        END  START
```

Глава 2. Таймеры и звук

2.1. Установка и чтение таймера

Все IBM PC используют микросхему таймера 8253 (или 8254) для согласования импульсов от микросхемы системных часов. Число циклов системных часов преобразуется в один импульс, а последовательность этих импульсов подсчитывается для определения времени или они могут быть посланы на громкоговоритель компьютера для генерации звука определенной частоты. Микросхема 8253 имеет три идентичных независимых канала, каждый из которых может программироваться.

Микросхема 8253 работает независимо от процессора. Процессор программирует микросхему и затем обращается к другим делам. Таким образом, микросхема 8253 действует как часы реального времени - она считает свои импульсы независимо от того, что происходит в компьютере. Однако максимальный программируемый интервал составляет приблизительно 1/12 с. Для подсчета интервалов времени в часы и минуты нужны какие-то другие средства. Именно по этой причине импульсы от нулевого канала микросхемы таймера накапливаются в переменной, находящейся в области данных BIOS. Этот процесс показан на рис. 2.1. Такое накопление обычно называется подсчетом времени суток. 18,2 раза в секунду выход канала 0 обрабатывается аппаратным прерыванием (прерыванием таймера), которое ненадолго останавливает процессор и увеличивает показания счетчика времени суток. Число 0 соответствует полночи 12:00; когда счетчик достигает значения, эквивалентного 24 часам, он сбрасывается на ноль. Другое время в течение суток легко определяется делением показателя счетчика на 18,2 для каждой

секунды. Счетчик времени суток используется в большинстве операций, связанных со временем.

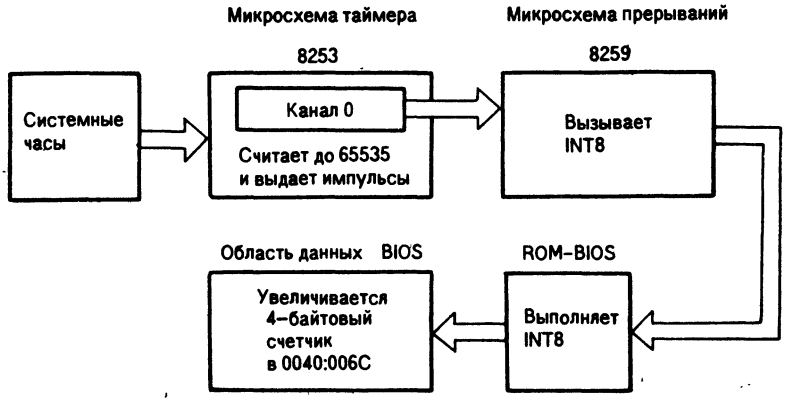


Рис. 2.1. Изменение счетчика времени суток BIOS

2.1.1. Программирование микросхемы таймера 8253/8254

Каждый из трех каналов микросхемы таймера 8253 (8254 для AT) состоит из трех регистров. Доступ к каждой группе из трех регистров осуществляется через один порт; номера портов от 40H до 42H соответствуют каналам 0 - 2. Порт связан с 8-битовым регистром ввода/вывода, который посылает и принимает данные для этого канала. Когда канал запрограммирован, через этот порт посылается 2-байтовое значение, сначала младший байт. Это число передается в 16-битовый регистр задвижки (latch register), который хранит это число и из которого копия помещается в 16-битовый регистр счетчика. В регистре счетчика число уменьшается на единицу каждый раз, когда импульс от системных часов пропускается через канал. Когда значение этого числа достигает нуля, канал выдает выходной сигнал и затем новая копия содержимого регистра задвижки передвигается в регистр счетчика, после чего процесс повторяется. Чем меньше число в регистре счетчика, тем быстрее ритм. Все три канала всегда активны: процессор не включает и не выключает их. Текущее значение любого из регистров счетчика может быть прочитано в любой момент времени, что не влияет на счет.

Каждый канал имеет две входные и одну выходную линии. Выходная линия выводит импульсы, возникающие в результате подсчета. Назначение этих сигналов варьируется в зависимости от типа IBM PC.

- Канал 0 используется системными часами времени суток. Он устанавливается BIOS при старте таким образом, что выдает импульсы приблизительно 18,2 раза в секунду. 4-байтовый счетчик этих импульсов хранится в памяти по адресу 0040:006C (младший байт хранится первым). Каждый импульс инициирует прерывание таймера (номер 8) и именно это прерывание увеличивает показание счетчика. Это аппаратное прерывание, поэтому оно обрабатывается всегда независимо от того, чем занят процессор, если только разрешены аппаратные прерывания (см. [1.2.2]). Выходная линия используется также для синхронизации некоторых дисковых операций, поэтому если вы изменили ее значение, то вам необходимо восстановить первоначальное значение перед обращением к диску.
- Канал 1 управляет обновлением памяти на всех машинах, кроме PCjr, поэтому его лучше не трогать. Выходная линия этого канала связана с микросхемой прямого доступа к памяти [5.4.2], и ее импульс заставляет микросхему DMA обновить всю память. На PCjr канал 1 служит для преобразования входных данных с клавиатуры из последовательной в параллельную форму. PCjr не использует микросхему прямого доступа к памяти, поэтому когда он вместо этого прогоняет данные через процессор, прерывание от таймера заблокировано. Канал 1 используется для подсчета заблокированных импульсов часов времени суток, с тем чтобы можно было обновить значение счетчика после завершения дисковых операций.
- Канал 2 связан с громкоговорителем компьютера и он производит простые прямоугольные импульсы для генерации звука. Программисты имеют больше возможностей управлять вторым каналом, чем остальными. Простые звуки могут генерироваться одновременно с другими программными операциями, а более сложные звуковые эффекты могут быть достигнуты за счет использования процессора. Канал 2 может быть отсоединен от громкоговорителя и служить для синхронизации. Наконец, выходная линия канала 2 связана с динамиком компьютера. Однако динамик не будет генерировать звук до определенных установок микросхемы интерфейса с периферией 8255.

Две входные линии для каждого канала состоят из линии часов, передающей сигнал от микросхемы системных часов и линии, называемой воротами (gate), которая включает и выключает сигнал от часов. Ворота всегда открыты для сигналов часов по каналам 0 и 1. Но они могут быть закрытыми для канала 2, что

позволяет производить некоторые специальные манипуляции со звуком. Ворота закрываются установкой младшего бита порта с адресом 61H, который является регистром микросхемы 8255; сброс этого бита снова открывает ворота. Эта микросхема обсуждается в [1.1.1]. Отметим, что, как и выход канала 2, бит 1 порта 61H связан с динамиком и также может использоваться для генерации звука. На рис. 2.2 приведена диаграмма микросхемы таймера 8253.

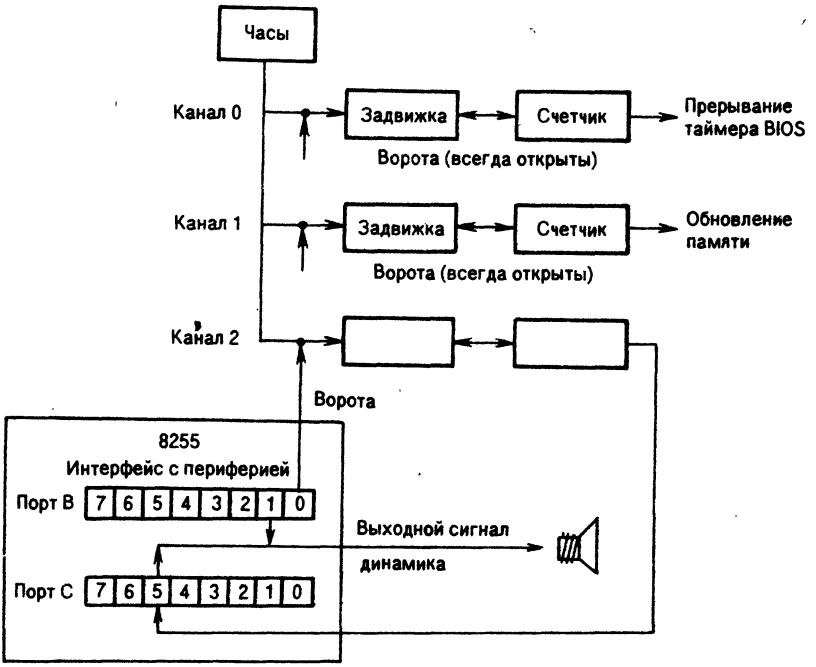


Рис. 2.2. Микросхема таймера 8253/8254

Микросхема таймера может применяться непосредственно для временных операций, но это редко бывает удобным. Ввод с часов производится 1,19318 млн. раз в секунду (даже на AT, где системные часы идут быстрее, микросхема таймера получает сигнал с частотой 1,19 МГц). Поскольку максимальное число, которое может храниться в 16 битах, равно 65535 и поскольку это число делится на частоту импульсов от часов, равную 18,2, то максимальный возможный интервал между импульсами равен приблизи-

тельно 1/12 с. Поэтому в большинстве временных операций используется счетчик времени суток BIOS. Для подсчета времени читается значение времени суток и сравнивается с некоторым ранее запомненным значением для определения числа импульсов, прошедших с того момента. Специальный способ, описанный в [2.1.7], позволяет использовать счетчик времени суток для операций в реальном времени.

Микросхема таймера 8253 предоставляет разработчикам оборудования 6 режимов работы для каждого канала. Программисты обычно ограничиваются третьим режимом как для канала 0 при синхронизации, так и для канала 2 при синхронизации или генерации звука. В этом режиме, как только регистр задвижки получает число, он немедленно загружает копию в регистр счетчика. Когда значение в счетчике достигает нуля, регистр задвижки мгновенно перезагружает счетчик и т.д. В течение первой половины отсчета выходная линия включена, а в течение второй - выключена. В результате получаются прямоугольные волны, которые одинаково пригодны как для генерации звука, так и для подсчета.

8-битовый командный регистр управляет способом загрузки чисел в канал. Адрес порта для этого регистра равен 43H. Командному регистру передается байт, который сообщает, какой канал программировать, в каком режиме, а также один или оба байта регистра задвижки должны быть переданы. Он показывает также, будет ли число в двоичной или BCD (двоично-кодированной десятичной) форме. Значение битов этого регистра таково:

бит	0	если 0, двоичные данные, иначе BCD
	3-1	номер режима, 1 - 5 (000 - 101)
	5-4	тип операции:
		00 = передать значение счетчика в задвижку
		01 = читать/писать только старший байт
		10 = читать/писать только младший байт
		11 = читать/писать старший байт, потом младший
	7-6	номер программируемого канала, 0 - 2 (00 -10)

Короче говоря, для программирования микросхемы 8253 надо выполнить три основных шага. После того как третий шаг завершен, запрограммированный канал немедленно начинает функционировать по новой программе.

1. Послать в командный регистр (43H) байт, представляющий цепочку битов, которые выбирают канал, статус чтения/записи, режим операции и форму представления чисел.

2. Для канала 2 надо разрешить сигнал от часов, установив в 1 бит 0 порта с адресом 61H. (Когда бит 1 этого регистра установлен в 1, канал 2 управляет динамиком. Сбросьте его в 0 для операций синхронизации.)

3. Вычислить значение счетчика от 0 до 65535, поместить его в AX и послать сначала младший, а затем старший байт в регистр ввода/вывода канала (40H - 42H).

Каналы микросхемы 8253 работают постоянно. По этой причине программы всегда должны восстанавливать начальные установки регистров 8253 перед завершением. В частности, если при завершении программы генерируется звук, то он будет продолжаться даже после того, как MS-DOS получит управление и загрузит другую программу. Имейте это в виду при написании процедуры выхода по «Ctrl-Break» [3.2.8].

Низкий уровень

В данном примере канал 0 программируется на другое значение, а не на установленное BIOS при старте. Причина изменения установки состоит в том, чтобы изменить интервал изменения счетчика времени суток на величину, большую, чем 18,2 раза в секунду. Частота обновления счетчика изменяется, скажем, на 1000 раз в секунду с целью проведения точных лабораторных измерений. Значение задвижки должно быть 1193 (1193180 тактов в секунду / 10000). Как читать текущее значение регистра счетчика см. в [2.1.8]. Перед дисковыми операциями оригинальное значение задвижки должно быть восстановлено, поскольку канал 0 используется для синхронизации дисковых операций. Максимально возможное значение - 65535 тактов часов между импульсами от канала - может быть достигнуто засылкой 0 в регистр задвижки (0 немедленно превращается в 65535 при уменьшении на единицу).

```
;---установка регистров ввода/вывода
COM_REG  EQU  43H                ;адрес командного регистра
CHAN_0   EQU  40H                ;адрес канала 0
         MOV  AL,00110110B       ;установка битов для канала 2
         OUT  COMMAND_REG,AL     ;засылка в командный регистр
;---посылка счетчика в задвижку
         MOV  AX,1193            ;счетчик для 100 импульсов/с
         OUT  CHANNEL_2,AL       ;посылка младшего байта
         MOV  AL,AH              ;готовим для посылки старший байт
         OUT  CHANNEL_2,AL       ;посылка старшего байта
```

2.1.2. Установка/чтение времени

При старте MS-DOS запрашивает у пользователя текущее время. Введенное значение помещается в 4 байта, хранящие счетчик времени суток (начиная с 0040:006C, младший байт хранится первым). Но сначала оно преобразуется в форму, в которой подсчитывается время суток, т.е. время преобразуется в число восемнадцатых долей секунды, прошедших с полуночи. Это число постоянно обновляется 18,2 раза в секунду прерыванием таймера. Когда появляется очередной запрос на время, текущее значение счетчика времени суток преобразуется обратно в привычный формат часы-минуты-секунды. Если при старте не было введено значение, то счетчик устанавливается в ноль, как будто сейчас полночь. Компьютеры, снабженные микросхемой календаря-часов, могут автоматически устанавливать счетчик времени суток.

Высокий уровень

TIME$ устанавливает или получает время в виде строки чч:мм:сс, где часы меняются от 0 до 23, начиная с полуночи. Для 5:10 дня:

```
100 TIME$ = "17:10:00"  'установка времени
110 PRINT TIME$         'вывод времени
```

Поскольку TIME$ возвращает строку, для выделения отдельных частей показания часов можно использовать строковые функции MID$, LEFT$ и RIGHT$. Например, чтобы преобразовать время 17:10:00 в 5:10, вы должны вырезать строку символов, соответствующую часам, преобразовать ее в числовой вид (с помощью функции VAL), вычесть 12, а затем представить результат снова в виде строки:

```
100 T$ = TIME$                          'получаем строку времени
110 HOUR$ = LEFT$(T$,2)                 'выделяем значение часов
120 MINUTES$ = MID$(T$,4,2)             'выделяем значение минут
130 NEWHOUR = VAL(HOUR$)                'преобразуем часы в число
140 IF NEWHOUR > 12 THEN NEWHOUR = NEWHOUR - 12
150 NEWHOUR$ = STR$(NEWHOUR)            'новое значение в строку
160 NEWTIME$ = NEWHOUR$ + ":" + MINUTES$ 'делаем новую строку
```

Средний уровень

MS-DOS предоставляет прерывания для чтения и установки времени, производя необходимые преобразования между значением счетчика времени суток и часами-минутами-секундами. Время выдается с точностью до 1/100 секунды, но поскольку счетчик времени суток обновляется с частотой в пять раз меньшей, показания сотых секунд очень приближенные. Функция 2CH прерывания 21H выдает время, а функция 2DH устанавливает его. В обоих случаях CH содержит часы (от 0 до 23, где 0 соответствует полночи), CL - минуты (от 0 до 59), DH - секунды (от 0 до 59) и DL - сотые доли секунд (от 0 до 99).

Кроме того, при получении времени функцией 2CH AL содержит номер дня недели (0 = воскресенье). Значение дня будет верным, только если была установлена дата. DOS вычисляет номер дня недели по дате. Отметим также, что при установке времени функцией 2DH AL отмечает правильность введенного значения времени (0 = правильно, FF = неправильно).

```
;---установка времени
MOV  CH,HOURS          ;вводим значения времени
MOV  CL,MINUTES
MOV  DH,SECONDS
MOV  DL,HUNDREDTHS
MOV  AH,2DH            ;номер функции установки времени
INT  21H               ;устанавливаем время
CMP  AH,0FFH           ;проверяем правильность значения
JE   ERROR             ;переход на обработку ошибки
;---получение времени
MOV  AH,2CH            ;номер функции получения времени
INT  21H               ;получаем время
MOV  DAY_OF_WEEK,AH    ;получаем день недели из AH
```

Низкий уровень

Если вы изменили скорость импульсов канала 1 микросхемы 8253 для специальных приложений, то необходимо написать свою процедуру декодирования показаний счетчика времени суток. BIOS позволяет диапазон значений счетчика от 0 до 1,573 миллиона и это может быть изменено только путем изменения прерывания таймера. Поэтому часы, реально показывающие сотые доли секунды, не могут работать 24 часа без специально написанной

программы. Отметим также, что байт 0040:0070 устанавливается в ноль при старте, а затем увеличивается на 1 (не больше) по ходу часов.

2.1.3. Установка/чтение даты

При включении компьютера MS-DOS запрашивает у пользователя текущие дату и время. Время записывается в области данных BIOS. Дата же содержится в переменной в COMMAND.COM. Она хранится в формате трех последовательных байтов, которые содержат соответственно день месяца, номер месяца и номер года, начиная с 0, где 0 соответствует 1980 году. В отличие от счетчика времени суток, адрес даты в памяти меняется с изменением версии DOS и положением в памяти COMMAND.COM. По этой причине для получения даты всегда надо использовать готовые утилиты Бейсика или MS-DOS, а не обращаться к этой переменной напрямую.

Машины, оборудованные микросхемой календаря-часов, автоматически устанавливают время и дату с помощью специальной программы (обычно запускаемой при старте через файл AUTOEXEC.BAT). Как получить доступ к микросхеме календаря-часов см. в [2.1.4]. Отметим также, что когда счетчик времени суток BIOS переходит через отметку 24 часов, MS-DOS меняет дату.

Высокий уровень

Оператор Бейсика DATE$ устанавливает или получает дату в виде строки формата ММ-ДД-ГГГГ. Можно использовать косую черту (/) вместо дефиса (-). Первые две цифры года могут быть опущены. Для 31 октября 1984 г.:

```
100 DATE$ = "10/31/84" 'установка даты
110 PRINT DATE$        'вывод даты
... и на дисплее будет выведено: 10-31-1984.
```

Средний уровень

Функции 2АН и 2ВН прерывания 21Н получают и устанавливают дату. Для получения даты поместите в АН 2АН и выполните прерывание. При возврате СХ будет содержать год в виде числа от 0 до 119, что соответствует диапазону лет 1980 - 2099

(можно сказать, что выдается смещение относительно 1980 года). DH содержит номер месяца, а DL - день.

```
MOV  AH,2AH        ;номер функции получения даты
INT  21H           ;получение даты
MOV  DAY,DL        ;день из DL
MOV  MONTH,DH      ;месяц из DH
ADD  CX,1980       ;добавляем базу к году
MOV  YEAR,CX       ;получаем номер года
```

Для установки даты поместите день, месяц и год в те же регистры и выполните функцию 2BH. Если значения, указанные для даты, неверны, то в AL будет возвращено FF, в противном случае - 0.

```
MOV  DL,DAY        ;помещаем день в DL
MOV  DH,MONTH      ;помещаем месяц в DH
MOV  CX,YEAR       ;помещаем год в CX
SUB  CX,1980       ;берем смещение относительно 1980
MOV  AH,2BH        ;номер функции установки даты
INT  21H           ;установка даты
CMP  AH,0FFH       ;проверяем успешность операции
JE   ERROR         ;неверная дата, идем на обработку ошибки
```

2.1.4. Установка/чтение часов реального времени

Часы реального времени имеют свой собственный процессор, который может подсчитывать время, не влияя на другие компьютерные операции. Они имеют также независимый источник питания, используемый, когда компьютер выключен. Программно можно как читать, так и устанавливать часы реального времени. Обычно имеется дополнительное программное обеспечение, которое устанавливает счетчик времени суток BIOS и переменную даты DOS таким образом, чтобы они соответствовали текущим показаниям часов реального времени. Но можно программно проверить соответствие между ними и при обнаружении разногласий принять необходимые меры.

Различные установки времени и даты осуществляются через набор адресов портов. Многие многофункциональные платы расширения для IBM PC имеют часы реального времени, но, к сожалению, нет стандартной микросхемы и диапазона адресов портов. AT оборудуется часами реального времени, основанными на микросхеме MC146818 фирмы Motorola, которые используют те

же регистры, что и микросхема, содержащая данные о конфигурации системы. Доступ к этим регистрам можно получить, послав сначала номер требуемого регистра в порт 70H, а затем прочитав значение регистра через порт 71H. Регистры, связанные с часами, следующие:

Номер регистра	Функция
00H	секунды
01H	секундная тревога
02H	минуты
03H	минутная тревога
04H	часы
05H	часовая тревога
06H	день недели
07H	день месяца
08H	месяц
09H	год
0AH	регистр статуса A
0BH	регистр статуса B
0CH	регистр статуса C
0DH	регистр статуса D

Биты четырех статусных регистров выполняют различные функции, из которых интерес для программистов могут представлять следующие:

Регистр A:	бит 7	1 = идет модификация времени (надо ждать значения 0, чтобы читать)
Регистр B:	бит 6	1 = разрешено периодическое прерывание
	бит 5	1 = разрешено прерывание тревоги
	бит 4	1 = разрешено прерывание конца модификации
	бит 1	1 = часы считаются до 24, 0 = до 12
	бит 0	1 = разрешено запоминание времени суток

Часы реального времени на AT могут вызывать аппаратное прерывание IRQ8. Программа может установить вектор этого прерывания на любую процедуру, которую требуется выполнить в определенное время [1.2.3]. Используйте вектор 4AH. Операции в реальном времени, производимые таким образом, менее хлопотны, чем обсуждаемые в [2.1.7] (хотя и ценой компактности программ).

Прерывание может вызываться одним из трех способов, каждый из которых запрещен при старте. Периодическое прерывание происходит через определенные интервалы времени. Периодичность приближенно равна одной миллисекунде. Прерывание тревоги происходит, когда значение трех регистров тревоги совпадает со значениями соответствующих временных регистров. Прерывание конца модификации происходит после каждого обновления значений регистров микросхемы.

Прерывание 1AH расширено в BIOS AT, чтобы оно позволяло читать и устанавливать часы реального времени. Поскольку показания никогда не состоят более чем из двух десятичных цифр, значения времени выдаются в двоично-кодированной десятичной форме (BCD), когда байт делится на две половины и каждая десятичная цифра представляется четырьмя битами. Такой формат позволяет легко переводить числа в форму ASCII. Программе нужно только сдвинуть половину байта в младший конец регистра и добавить 48 для получения кода ASCII, соответствующего данному числу. Для всех IBM PC функции 0 и 1 прерывания 1AH читают и устанавливают счетчик времени суток BIOS. Для часов реального времени AT имеется шесть новых функций.

Функция 2: Чтение времени из часов реального времени
При возврате: CH = часы в BCD
CL = минуты в BCD
DH = секунды в BCD

Функция 3: Установка времени часов реального времени
При входе: CH = часы в BCD
CL = минуты в BCD
DH = секунды в BCD
DL = if daylight savings, else 1

Функция 4: Чтение даты из часов реального времени
При возврате: CH = век в BCD (19 или 20)
CL = год в BCD (с 1980)
DH = месяц в BCD
DL = день месяца в BCD

Функция 5: Установка даты часов реального времени
При входе: CH = век в BCD (19 или 20)
CL = год в BCD (с 1980)
DH = месяц в BCD
DL = день месяца в BCD

Функция 6: Установка тревоги для часов реального времени
При входе: CH = часы в BCD
CL = минуты в BCD
DH = секунды в BCD

Функция 7: Сброс тревоги (нет входных регистров)

Тревога устанавливается как смещение относительно текущего момента времени. Максимальный период равен 23:59:59. Как уже говорилось выше, вектор прерывания 4AH должен указывать на процедуру обработки тревоги. Отметим, что если часы не работают (наиболее вероятно, из-за отсутствия питания), то выполнение функций 2, 4 и 6 устанавливает флаг переноса.

2.1.5. Задержка программных операций

Если вы осуществляете задержку в программе посредством пустого цикла, то вам может потребоваться много времени для того, чтобы добиться нужного времени задержки. Даже если вы определите требуемую длительность, то нельзя быть уверенным, что ваша программа будет давать нужное время задержки при всех условиях. Длительность цикла может меняться в зависимости от применяемого компилятора (или, для Бейсика, от того, компилируется программа или нет). А в наше время, когда имеется большой набор машин, совместимых с IBM PC (с широким диапазоном скорости процессора), даже цикл на языке ассемблера может приводить к различным временам задержки. Поэтому разумно определять время программной задержки непосредственно по часам. Частота отсчета 18,2 раза в секунду, используемая для модификации счетчика времени суток, должна вполне удовлетворять большинству потребностей (как увеличить частоту отсчетов см. в [2.1.1]).

Чтобы обеспечить задержку данной продолжительности, программа должна подсчитать требуемое число импульсов счетчика времени суток. Это значение добавляется к считанному текущему значению счетчика. Затем программа постоянно считывает значение счетчика и сравнивает его с запомненным. Когда достигается равенство, требуемая задержка прошла и можно продолжать выполнение программы. Четыре байта, в которых содержится значение счетчика времени суток, хранятся, начиная с адреса 0040:006C (как обычно, младший байт имеет меньший адрес). Для задержек меньше 14 с можно пользоваться только младшим байтом. Два младших байта позволяют задержки до одного часа (точнее, на полсекунды меньше, чем час).

Высокий уровень

В Бейсике можно использовать оператор SOUND [2.2.2] со значением частоты, равным 32767. В этом случае звук не будет генерироваться вообще. Это отсутствие звука будет длиться

столько отсчетов времени суток, сколько вы укажете. Для 5-секундной задержки нужен 91 отсчет (5 x 18,2). Поэтому

```
100 SOUND 32767,91      'останавливает программу на 5 с
```

Для прямого чтения счетчика времени суток нужно:

```
100 DEF SEG = 0                          'установка сегмента на начало памяти
110 LOWBYTE = PEEK(&H46C)                'получение младшего байта
120 NEXTBYTE = PEEK(&H46D)               'получение следующего байта
130 LOWCOUNT = NEXTBYTE*256 + LOWBYTE 'значение двух байтов
```

Средний уровень

Прочитайте значение счетчика времени суток BIOS, используя функцию 0 прерывания 1AH, и добавьте к нему необходимое число импульсов по 1/18 с. После этого считывайте текущие значения счетчика времени суток, постоянно сравнивая с требуемой величиной. При достижении равенства надо заканчивать задержку. Прерывание 1AH возвращает два младших байта в DX (большинство задержек укладываются в этих пределах), поэтому два старших байта, возвращаемые в CX, могут игнорироваться, что позволит вам избежать 32-байтовых операций. В данном примере установлена задержка на 5 с, что соответствует 91 отсчету:

```
;---получение значения счетчика и установка задержки
        MOV AH,0        ;номер функции для чтения
        INT  1AH        ;получаем значение счетчика
        ADD DX,91       ;добавляем 5 с к младшему слову
        MOV BX,DX       ;запоминаем требуемое значение в BX
;---постоянная проверка значения счетчика времени суток BIOS
REPEAT: INT  1AH        ;получаем значение счетчика
        CMP DX,BX       ;сравниваем с искомым
        JNE  REPEAT     ;если не равен, то повторяем снова
                        ;иначе - задержка окончена
```

AT имеет добавочную функцию прерывания 15H, которая позволяет осуществить задержку на указанное время. Поместите 86H в AH, а число микросекунд задержки - в CX:DX. После этого выполните прерывание.

2.1.6. Операции, запрограммированные во времени

Программа определяет время для выполнения определенной операции точно так же, как и человек: берется начальное показание счетчика времени суток и затем сравнивается с последующими показаниями. Можно получать значения в формате часы-минуты-секунды, но слишком хлопотно вычислять разницу между такими показаниями, поскольку система счета не десятичная. Лучше прямо читать счетчик времени суток BIOS, измерять продолжительность в 1/18 с, а затем уже переводить ее в обычный формат чч:мм:сс.

```
100 GOSUB 500                          'получаем значение счетчика
110 START = TOTAL                      'сохраняем начальное значение в
START
. (далее идет процесс, длительность которого измеряется)

300 GOSUB 500                          'получаем финальное значение
310 TOTAL = TOTAL - START              'подсчитываем число импульсов
320 HOURS = FIX(TOTAL/65520)           'вычисляем число часов
330 TOTAL = TOTAL - HOURS*65520        'вычитаем часы из TOTAL
340 MINUTES = FIX(TOTAL/1092)          'вычисляем число минут
350 TOTAL = TOTAL - MINUTES*1092       'вычитаем минуты из TOTAL
360 SECONDS = FIX(TOTAL/18.2)          'вычисляем число секунд
370 PRINT HOURS,MINUTES,SECONDS        'печатаем результат
380 END

500 DEF SEG = 0                        'подпрограмма чтения времени суток
510 A = PEEK(&H46C)                    'получаем младший байт
520 A = PEEK(&H46D)                    'получаем следующий байт
530 A = PEEK(&H46E)                    'и еще один
540 TOTAL = A + B*256 + C*65535        'подсчитываем результат в TOTAL
550 RETURN                             'все сделано
```

Функция TIMER в Бейсике возвращает число секунд, прошедших с момента, когда счетчик времени суток был последний раз установлен в 0. Обычно это число секунд, прошедших со времени последнего включения компьютера. Если при старте системы системное время было установлено правильно, то TIMER возвращает число секунд, прошедших с полуночи. Просто напишите N = TIMER.

Средний уровень

Прерывание 1AH имеет две функции для установки (AH = 1) и получения (AH = 0) счетчика времени суток. Для чтения счетчика надо просто выполнить прерывание с AH = 0. При возврате значение счетчика содержится в CX:DX, причем младшее слово в CX. AL содержит 0, если счетчик не переходил через границу 24 часов с момента последней установки. Для установки счетчика поместите два слова в те же регистры, а в AH - 1. В приведенном примере измеряются промежутки времени в пределах часа. При этом нужны только два младших байта счетчика. Но в этом случае необходимо проверять, не было ли перехода через границу, когда начальное значение было больше, чем следующее.

```
;---в сегменте данных
OLDCOUNTDW   0                  ;храним начальное значение счетчика
;---получаем начальное значение счетчика
             MOV  AH,0          ;номер функции
             INT  1AH           ;получаем значение счетчика
             MOV  OLDCOUNT,DX   ;сохраняем начальное значение

(здесь идет процесс, длительность которого измеряется)
;---позднее вычисляем длительность процесса
             MOV  AH,0          ;номер функции
             INT  1AH           ;получаем значение счетчика
             MOV  BX,OLDCOUNT   ;считываем старое значение
             CMP  BX,DX         ;проверяем на переполнение
             JG   ADJUST        ;обработка переполнения
             SUB  DX,BX         ;иначе - берем разность
             JMP  SHORT FIG_TIME ;и переводим ее в обычный вид
;---обработка переполнения
ADJUST:      MOV  CX,0FFFFH     ;помещаем в CX максимальное число
             SUB  CX,BX         ;вычитаем первое значение
             ADD  CX,DX         ;добавляем второе значение
             MOV  DX,CX         ;результат храним в DX
;---процедура перевода времени в обычный формат
FIG_TIME:                       ;делим на 18,2 с и т.д.
```

2.1.7. Управление работой в реальном времени

При операциях в реальном времени программа выполняет инструкции в указанный момент времени, а не при первой возможности. Такого рода операции обычно ассоциируются с робототехникой, но существует множество других приложений. Имеется

несколько подходов к операциям в реальном времени. Для программ, которые не должны ничего делать в промежутке между инструкциями, требующими временной привязки, можно просто периодически проверять счетчик времени суток, ожидая наступления нужного момента. Такой подход практически сводится к набору пустых циклов, описанных в [2.1.5].

Второй подход более сложен. Он используется, когда программа постоянно занята какой-либо работой, но должна в определенные моменты времени прерывать свои операции для выполнения конкретной задачи. В таком случае расширяют прерывание таймера, которое выполняется 18,2 раза в секунду. Когда это прерывание происходит, дополнительный код проверяет новое значение счетчика времени суток, и если наступил соответствующий момент, запускает нужную процедуру. Схема процесса показана на рис. 2.3. Приведенные здесь простые примеры демонстрируют, как создать в своей программе будильник, который устанавливается пользователем и подает звуковой сигнал, когда подошло время. (Более сложный пример низкого уровня в [2.2.6] исполняет музыку, в то время как процессор занят другими делами.)

Высокий уровень

Бейсик обеспечивает примитивный контроль над операциями в реальном времени посредством оператора ON TIMER(n) GOSUB. Когда программа встречает этот оператор, она начинает отсчитывать n секунд. Тем временем выполнение программы продолжается. Когда n секунд прошло, программа переходит на подпрограмму, начинающуюся с указанного номера строки, выполняет ее и возвращает управление на то место, откуда была вызвана подпрограмма. После этого отсчет снова начинается с нуля и подпрограмма будет вызвана повторно еще через n секунд.

ON TIMER не будет функционировать до тех пор, пока он не разрешен оператором TIMER ON. Оператор TIMER OFF запрещает его работу. В тех случаях, когда отсчет времени необходимо продолжить, но переход на подпрограмму должен быть задержан, надо использовать оператор TIMER STOP. В этом случае отмечается, что n секунд прошло, но переход на подпрограмму будет выполнен только после того, как встретится оператор TIMER ON.

Поскольку он повторяется, оператор ON TIMER особенно полезен для вывода на экран текущего времени:

```
100 ON TIMER(60) GOSUB 500      'меняем показания часов каждые 60
110 TIMER ON                    'секунд и разрешаем работу таймера
```

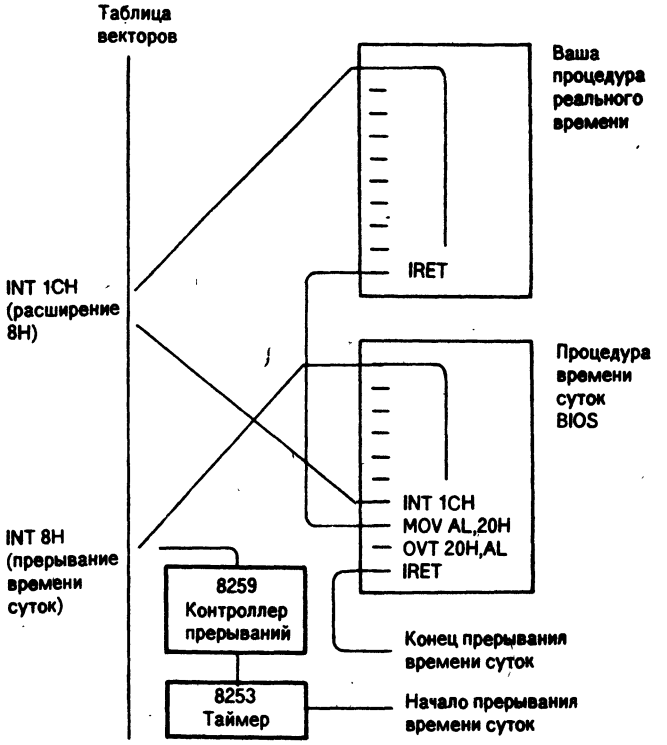


Рис. 2.3. Расширение прерывания таймера

```
500 LOCATE 1,35:PRINT "TIME: ";LEFT$(TIME$,5) 'позиционируем
510 RETURN                                     'курсор и печатаем время
```

Низкий уровень

BIOS содержит специальное пустое прерывание (1CH), которое ничего не делает, пока вы не напишете для него процедуру. При старте вектор этого прерывания указывает на инструкцию IRET (возврат из прерывания); при его вызове происходит моментальный возврат. Но прерывание 1CH интересно тем, что оно вызыва-

ется прерыванием таймера BIOS после того, как это прерывание обновило значение счетчика времени суток. Можно сказать, что это аппаратное прерывание, происходящее автоматически 18,2 раза в секунду. Вы можете изменить вектор этого прерывания так, чтобы он указывал на процедуру в вашей программе. После этого ваша процедура будет вызываться 18,2 раза в секунду. О том, как написать и установить свою процедуру обработки прерывания, см. в [1.2.3].

Написанная вами процедура должна прочитать только что модифицированное значение счетчика времени суток, сравнить его с ожидаемым временем и выполнить то, что требуется, когда ожидаемое время, наконец, наступит. Естественно, что когда время еще не подошло, процедура просто возвращает управление, ничего не делая. Таким образом, процессор не выполняет лишней работы.

В приведенном примере процедура (не показанная здесь) запрашивает у пользователя число минут (до 60), которое должно пройти до того, как раздастся звонок будильника. Это число, запасенное в MINUTES, умножается на 1092 для перевода в эквивалентное число импульсов счетчика времени суток. Для периода в пределах одного часа достаточно 16 бит - более длинные периоды требуют более сложных 32-битовых операций. Это число импульсов добавляется к младшему слову текущего значения счетчика времени суток и запоминается в ALARMCOUNT.

Затем вектор прерывания 1CH изменяется таким образом, чтобы он указывал на процедуру ALARM. Помните, что как только вектор будет изменен, ALARM будет автоматически вызываться 18,2 раза в секунду. При вызове эта процедура читает текущее значение счетчика времени суток через прерывание 1AH и сравнивает с ALARMCOUNT. При совпадении этих величин вызывается процедура BEEP (также не показанная здесь, см. [2.2.4]), которая выдает звуковой сигнал. В противном случае происходит возврат. Обычный код возврата из аппаратных прерываний (MOV AH,20H / OUT 20H,AL) включать в процедуру не нужно, так как он будет в прерывании таймера. Будьте внимательны и не забудьте сохранить изменяемые регистры.

```
;---в сегменте данных
MINUTES   DW   0                ;хранит число минут до звонка
ALCOUNT   DW   0                ;хранит счетчик времени для звонка
;---установка ожидаемого значения счетчика времени суток
          CALL REQ_MIN          ;запрос числа минут до звонка
          MOV  AX,MINUTES       ;пересылка в AX
          MOV  BX,1092          ;число импульсов счетчика в минуте
          MUL  BX               ;умножаем - результат в AX
```

```
;получаем текущее значение счетчика
            MOV  AH,0                ;номер функции чтения счетчика
            INT  1AH                 ;читаем значение, младший байт в DX
                                     ;складываем оба значения
            ADD  AX,DX
            MOV  ALCOUNT,AX          ;получаем нужное значение счетчика
;---заменяем вектор пустого прерывания
            PUSH DS                  ;сохраняем сегмент данных
            MOV  AX,SEG ALARM        ;берем сегмент процедуры ALARM
            MOV  DS,AX               ;помещаем его в DS
            MOV  DX,OFFSET ALARM     ;берем смещение процедуры
            MOV  AL,1CH              ;номер изменяемого вектора
            MOV  AH,25H              ;функция изменения вектора
            INT  21H                 ;меняем вектор
            POP  DS                  ;восстанавливаем сегмент данных
;
;---дальше продолжается программа
;
;---в конце программы возвращаем вектор прерывания
            MOV  DX,0FF53H           ;оригинальные значения для
            MOV  AX,0F000H           ;прерывания 1CH
            MOV  DS,AX               ;помещаем сегмент в DS
            MOV  AL,1CH              ;номер изменяемого вектора
            MOV  AH,25H              ;номер функции
            INT  21H                 ;восстанавливаем вектор
;---процедура выдачи звукового сигнала
ALARM       PROC FAR                 ;создаем далекую процедуру
            PUSH AX                  ;сохраняем изменяемые регистры
            PUSH CX
            PUSH DX
;---читаем счетчик времени суток
            MOV  AH,0                ;номер функции чтения счетчика
            INT  1AH                 ;читаем значение счетчика
;---сравниваем с требуемым значением
            MOV  CX,ALCOUNT          ;берем требуемое значение
            CMP  DX,CX               ;сравниваем с текущим
            JNE  NOT_YET             ;если не равны, то на выход
;---выдаем звуковой сигнал, если значения совпали
            CALL BEEP                ;эта процедура не показана
```

```
;---иначе - возвращаемся из прерывания
NOT_YET: POP  DX                ;восстанавливаем регистры
         POP  CX
         POP  AX
         IRET                   ;возврат из прерывания
ALARM    ENDP                   ;конец процедуры
```

2.1.8. Генерация случайных чисел с помощью микросхемы таймера

Для генерации последовательности случайных чисел требуются сложные математические манипуляции. Но иногда программе в определенный момент требуется только одно случайное число, которое может быть получено просто чтением из канала микросхемы таймера. Бейсик использует это число в качестве ядра, по которому генерируется случайная последовательность. Конечно, вы не можете использовать ряд последовательно считанных значений в качестве случайной последовательности, так как сами по себе интервалы времени между считываниями будут неслучайными.

Высокий уровень

Бейсик имеет генератор случайных чисел, который может быть перезапущен с помощью оператора TIMER, с тем чтобы при каждом запуске программы генерировалась другая последовательность случайных чисел. Напишите RANDOMIZE TIMER, а затем используйте функцию RND для получения случайных чисел.

```
100 RANDOMIZE TIMER             'сброс генератора случайных чисел
110 PRINT RND,RND,RND           'печать трех случайных чисел
в результате получаем: 0,7122483 0,4695052 0,9132487
```

Низкий уровень

Поскольку регистр счетчика канала таймера перезагружается снова и снова данным числом (а в промежутках идет счет вниз до 0), выберите в качестве загружаемого в счетчик значения число, равное требуемому диапазону случайных чисел. Например, для получения случайного значения часа дня загружайте в счетчик 23.

Лучше всего использовать режим 3 канала 2 (порт 42H) микросхемы таймера [2.1.1]. Сначала установите для счетчика желаемый диапазон случайных чисел (в примере используется 10000, что приводит к выдаче случайного числа в диапазоне от 0

до 9999). Затем, чтобы получить из канала случайное число, надо подать команду командному регистру микросхемы таймера через порт 43H перенести текущее значение счетчика в регистр "задвижки", для чего надо сбросить биты 4 и 5. Этот перенос в регистр задвижки не мешает продолжающемуся счету. Затем установите оба бита 4 и 5 командного регистра, чтобы процессор мог читать из регистра задвижки. После этого две инструкции IN дадут сначала младший, а затем старший байт в регистре AL. Наконец, восстановите первоначальное значение регистра задвижки, чтобы счет продолжался в пределах указанного диапазона времени.

```
;---установка адресов портов
COM_REG    EQU  43H              ;адрес командного регистра
CHANNEL_2 EQU  42H               ;адрес канала 2
           CALL SET_COUNT        ;установка диапазона
.
;---здесь программа работает, а затем требует случайное число
.
           CALL GET_NUMBER       ;получение случайного числа
.
.
;---начинаем отсчет канала 2
SET_COUNT PROC
           MOV  AL,10110110B     ;канал 2, режим 2, оба байта
           OUT  COM_REG,AL       ;посылаем в командный регистр
           MOV  AX,10000         ;значение счетчика
           OUT  CHANNEL_2,AL     ;посылаем младший байт
           MOV  AL,AH            ;передвигаем старший байт в AL
           OUT  CHANNEL_2,AL     ;посылаем старший байт
           RET
SET_COUNT ENDP
;---получение случайного числа
READ_NUM  PROC
;---пересылаем значение счетчика в регистр задвижки
           MOV  AL,10000110B     ;требуемая команда
           OUT  COM_REG,AL       ;посылаем в командный регистр
;---читаем значение счетчика
           MOV  AL,10110110B     ;запрос на чтение/запись
           OUT  COM_REG,AL       ;посылаем запрос
           IN   AL,CHANNEL_2     ;получаем младший байт
           MOV  AH,AL            ;временно храним его в AH
           IN   AL,CHANNEL_2     ;получаем старший байт
           CALL SET_COUNT        ;восстанавливаем задвижку
           SWAP AH,AL            ;ставим байты на место
```

```
        RET                     ;теперь случайное число в AX
READ_NUM ENDP
```

2.2. Создание звука

Бейсик оснащен достаточно изощренными средствами для генерации звука, однако операционная система позволяет только просто подать звуковой сигнал. Если вы хотите получить какие-либо сложные звуки, то вы должны программировать микросхему таймера 8253. Канал 2 этой микросхемы прямо связан с динамиком компьютера. Когда этот канал программируется в режиме 3, он посылает прямоугольные волны данной частоты. Из-за простоты динамика он сглаживает края прямоугольной волны, получая более приятную для слуха синусоидальную волну. К сожалению, микросхема 8253 не меняет амплитуду волны, поэтому мы не можем менять громкость звука, издаваемого динамиком.

Динамик имеет не один, а два входа для генерации звука. На рис. 2.2 в [2.1.1] показано, что, кроме микросхемы таймера, сигнал посылает также микросхема интерфейса с периферией 8255 [1.1.1]. Частота импульсов каждой микросхемы может быть изменена, поэтому, комбинируя воздействия этих двух источников, мы можем получать специальные звуковые эффекты.

Только PCjr имеет специальную микросхему, управляющую генератором звука. Он может одновременно выдавать три разных тона плюс шум для звуковых эффектов. Громкость каждого из трех каналов может устанавливаться независимо. Другой уникальной возможностью PCjr является то, что он может управлять внешним источником звука, таким, как кассетный магнитофон.

2.2.1. Программирование генератора звука 76496 (только PCjr)

PCjr снабжен 4-канальным генератором звука, в котором три канала генерируют тона, а четвертый служит для генерации шума для звуковых эффектов. Все четыре канала программируются независимо, причем каждый имеет свой регулятор громкости, а затем выход с них объединяется в единый звуковой сигнал. Используется микросхема комплексного генератора звука TI SN76496N. Она имеет 8 регистров, по 2 для каждого канала, и все они адресуются через один порт с адресом 0C0H. Этот порт

служит только для записи; если подать инструкцию IN, то вся система будет заморожена.

PCjr имеет также разъем для внешнего источника звука. При старте системы звуковой канал получает выходной сигнал от микросхемы таймера 8253. Но этот канал может быть переключен на микросхему генератора звука или любой из двух внешних звуковых входов. Это достигается изменением битов 5 и 6 порта B микросхемы интерфейса с периферией 8255 (адрес порта 61H, см. [1.1.1]). Значение битов следующее:

Биты 6 и 5	Выбранная функция
00	микросхема таймера 8253
01	вход с кассетного магнитофона
10	вход канала ввода/вывода
11	генератор звука 76496

Для выбора источника звука в BIOS PCjr добавлена функция 80H прерывания 1AH. Поместите в AL номер кода от 0 до 3 в соответствии с приведенной выше таблицей, и вызовите функцию. Возвращаемых регистров нет. Генератор звука 76496 должен использовать этот звуковой канал, поскольку он не может управлять внутренним динамиком PCjr.

В общем случае, когда байт данных посылается генератору звука, биты 4-6 содержат код идентификации, сообщающий, какому из восьми регистров предназначены данные. Эти коды такие:

Биты 6-4	Адресуемый регистр
000	частота первого тона
001	громкость первого тона
010	частота второго тона
011	громкость второго тона
100	частота третьего тона
101	громкость третьего тона
110	частота четвертого тона
111	громкость четвертого тона

В случае регистров частоты тонов требуются два байта. Значение битов при этом следующее:

байт 1:	биты	0-3	младшие 4 бита частоты
		4-6	код идентификации регистра

		7	всегда равен 1
байт 2:	биты	0-5	старшие 6 бит частоты
		6	не используется
		7	всегда равен 0

Для установки частоты тона в регистр посылается 10-битовое значение, которое после деления на 111843 дает желаемую частоту в герцах. Таким образом, доступны частоты, начиная со 110 Гц и больше. (111843/2^10). Как только регистр инициализирован (и соответственно установлен порт B микросхемы 8255), немедленно начинается звуковой сигнал и продолжается до тех пор, пока он не будет прекращен. Не обязательно для изменения частоты посылать новые два байта. Если послан только второй байт (старшие 6 бит частоты), то он автоматически заменяет соответствующие данные в канале, к которому была последняя адресация. Эта возможность позволяет плавно варьировать частоту.

Генератору шума для программирования нужен только один байт. Значение битов для него следующее:

биты 0-1	плотность шума
2	качество шума
3	не используется
4-6	код идентификации регистра
7	всегда установлен в 1

Качество шума устанавливается на белый шум (постоянное шипение), когда бит 2 равен 1, и на периодический шум (волны звука), когда бит 2 равен 0. Плотность звука увеличивается при увеличении битов 0-1 от 00B до 10B; когда они установлены в 11B, звук меняется в зависимости от выходного тона канала 3.

Громкость каждого из четырех каналов изменяется в результате ослабления основного сигнала. Для этой установки требуется только один байт. Значение его битов следующее:

биты 0-3	ослабление сигнала
4-6	код идентификации регистра
7	всегда установлен в 1

Когда все 4 бита данных равны 0, громкость максимальна. Когда все они равны 1, звук полностью подавляется. Для получения звука промежуточной громкости может быть использована любая комбинация битов. Бит 0 ослабляет звук на 2 Дб (децибела), бит 1 на 4 Дб, бит 2 - на 8 Дб и бит 3 - на 16 Дб. Максимальное ослабление равно 28 Дб.

2.2.2. Генерация тона

Этот параграф объясняет, как производить звук, когда компьютер не занят ничем другим; в [2.2.3] показано, как это сделать, когда производятся другие действия. Забавно, но для программистов на ассемблере последнее проще. Для этого достаточно запрограммировать микросхему таймера 8253, которая работает независимо от процессора. В приведенном здесь методе процессор непосредственно управляет динамиком, поэтому программе приходится выполнять работу, которую может выполнять микросхема таймера. Хотя этот способ более труден, он допускает существенно больший контроль за динамиком, и создание большинства специальных звуковых эффектов (см. [2.2.8]) основывается на нем.

Высокий уровень

Оператор Бейсика SOUND используется для генерации тона в широком диапазоне частот и длительностей. Частота дается в герцах (от 37 до 32767), а длительность в импульсах счетчика времени суток BIOS (от 0 до 65535), причем в секунду происходит 18,2 импульса. SOUND 440,91 воспроизводит ноту A в течение 5 с (5 x 18,2). Частоты первой октавы, начиная с ноты C(до), таковы:

C(до)	523,3
D(ре)	587,3
E(ми)	659,3
F(фа)	698,5
G(соль)	784,0
A(ля)	880,0
B(си)	987,7

Частоты на октаву выше можно получить, удваивая эти значения, на две октавы выше - еще раз удваивая частоты. И наоборот, частоты на октаву ниже равны приблизительно половине этих значений (хорошо настроенное пианино точно не следует арифметическим интервалам).

Благодаря своему генератору звука [2.2.1] PCjr может использовать оператор SOUND для трех независимых каналов звука, причем громкость каждого из них управляема. В этом случае формат оператора: SOUND частота, длительность, громкость, канал. Громкость может меняться от 0 до 15, по умолчанию - 8. Номер канала может меняться от 0 до 2, по умолчанию - 0.

Поскольку PCjr может использовать возможности многоголосия и контроля звука только для внешнего динамика, надо сначала разрешить этот динамик. Это делается с помощью оператора SOUND ON. SOUND OFF передает контроль внутреннему динамику. Чтобы сыграть аккорд D-минор (ре-минор) (D-F-A) с малой громкостью, напишите:

```
100 SOUND ON                    'разрешение внешнего динамика
110 SOUND 587,50,3,0            'нота ре
120 SOUND 699,50,3,1            'нота фа
130 SOUND 880,50,3,1            'нота ля
```

Низкий уровень

Генерация звука с помощью адаптера интерфейса с периферией 8255 состоит во включении и выключении с желаемой частотой бита порта B, который связан с динамиком (бит 1). Порт B имеет адрес 61H (хотя AT не имеет микросхемы интерфейса с периферией 8255 как таковой, он использует для этой цели тот же адрес порта и тот же бит). Если программа переключает значение бита с максимально возможной частотой, то частота слишком высокая, чтобы быть полезной. Поэтому между двумя переключениями надо вставлять пустой цикл. Помните, что бит 0 порта B управляет воротами канала 2 микросхемы таймера, который в свою очередь связан с динамиком. Поэтому этот бит сбрасывается, отсоединяясь от канала таймера. На рис. 2.4 показано, как этот метод позволяет установить частоту звука.

В следующем примере введены две переменные. Одна, обозначенная "FREQUENCY", используется в качестве счетчика в пустом цикле между действиями включения и выключения. Чем меньше ее значение, тем быстрее происходит изменение бита и тем больше частота. Переменная же "NUMBER_CYCLES" устанавливает продолжительность тона. Она говорит, сколько раз должен быть повторен процесс включения и выключения. Чем больше это число, тем дольше звучит данный звук.

Отметим, что для этой процедуры аппаратные прерывания должны быть запрещены. Причина этого в том, что прерывание таймера происходит с такой частотой и регулярностью (18,2 раза в секунду), что оно будет существенно влиять на частоту. Имейте в виду, что пока прерывания запрещены, счетчик времени суток BIOS не будет работать. Если затем прочитать его значение, то оно будет отличаться на некоторую величину от реального до тех пор, пока не будет сделано соответствующее изменение.

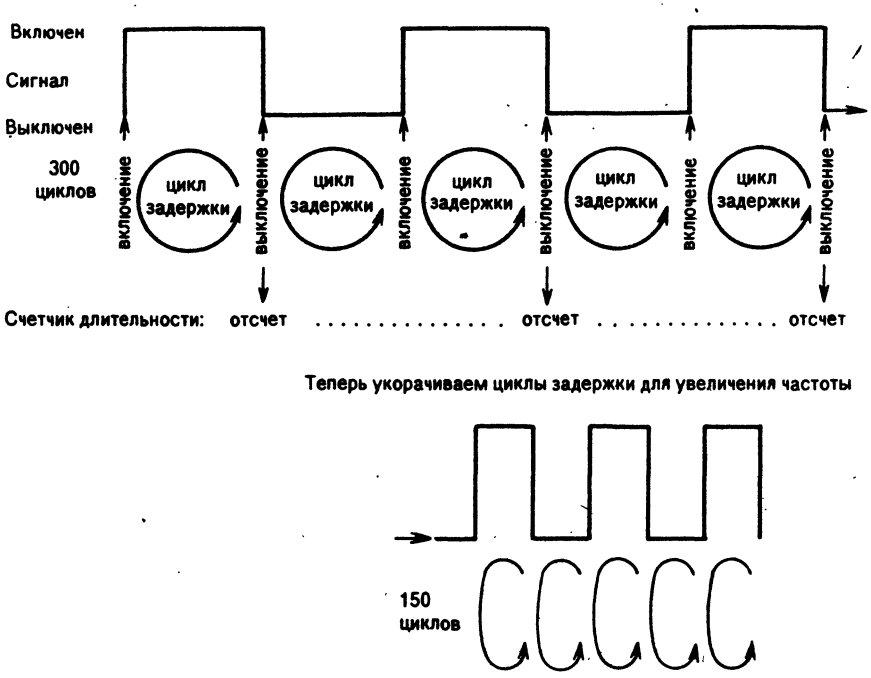


Рис. 2.4. Генерация звука микросхемой 8255

```
NUM_CES   EQU  1000
FREQ      EQU  300
PORT_B    EQU  61H
          CLI                ;запрет прерываний
          MOV  DX,NUM_CES    ;длительность тона в DX
          IN   AL,PORT_B     ;получаем значение из порта B
          AND  AL,11111110B  ;отключаем динамик от таймера
NEXT_CLE: OR   AL,00000010B  ;включаем динамик
          OUT  PORT_B,AL     ;посылаем команду в порт B
          MOV  CX,FREQ       ;задержка на полцикла в CX
FIR_HALF: LOOP FIR_HALF      ;делаем задержку
          AND  AL,11111101B  ;выключаем динамик
          OUT  PORT_B,AL     ;посылаем команду в порт B
          MOV  CX,FREQ       ;задержка на полцикла в CX
```

```
SEC_HALF: LOOP SEC_HALF    ;делаем задержку
          DEC  DX          ;вычитаем единицу из счетчика
          JNZ  NEXT_CLE    ;если 0, то надо заканчивать
          STI              ;разрешаем прерывания
```

2.2.3. Генерация звука одновременно с другими действиями

Для программистов на Бейсике различие между этим и предыдущим параграфом несущественно. Но программисты на ассемблере должны использовать совершенно иной метод. Поскольку микросхема таймера 8253 работает независимо от процессора, очень легко генерировать звук, который издается одновременно с выполнением других операций. Вы должны просто запрограммировать канал 2 этой микросхемы для генерации определенной частоты, а затем перепрограммировать микросхему для выключения звука.

Высокий уровень

Оператор SOUND в Бейсике не позволяет генерировать звук одновременно с другими действиями, но оператор PLAY, если ему это задать, позволяет. За оператором PLAY должна следовать строка, которая сообщает, какие ноты должны быть сыграны, какой длительности, а также другие характеристики. Детали командной строки PLAY обсуждаются в [2.2.5]. Если строка содержит буквы MB (фоновая музыка), то она помещается в специальный буфер и выполняется одновременно с другими программными действиями. Напротив, MF (музыка на переднем плане) останавливает все программные операции до тех пор, пока не будет исполнена вся строка. Вот как воспроизвести исполнение одной ноты A (ля) в фоновом режиме:

```
100 PLAY "MB A"          'исполняется нота ля...
110 .....                'и следующие операторы программы
```

Отметим, что в фоновом режиме оператор X = PLAY(0) возвращает число нот (до 32), которое осталось сыграть. В многоканальном режиме на PCjr возвращается число нот в буфере данного канала (0-2), номер которого указан в скобках.

Низкий уровень

Пошлите счетчик в канал 2, как объяснено в [2.1.1]. Микросхема должна быть предварительно разрешена через порт B микро-

схемы интерфейса с периферией 8255 (адрес 61H). Вычислите требуемое значение счетчика для задвижки, разделив 1,19 миллиона на требуемую частоту в герцах. Звук будет продолжаться до тех пор, пока не будут закрыты ворота канала 2. Поэтому вы должны сбросить бит 1 порта B в 0, иначе звук будет продолжаться бесконечно и может быть прекращен только при перезагрузке компьютера. Для точного регулирования длительности звука можно использовать счетчик времени суток BIOS (см. [2.1.6]). В данном примере генерируется частота 440 Гц. Звук прекращается после нажатия любой клавиши на клавиатуре.

```
;---разрешение канала 2 установкой порта B микросхемы 8255
PORT_B   EQU 61H            ;установка адреса порта B
         IN   AL,PORT_B     ;чтение его значения
         OR   AL,3          ;установка двух младших битов
         OUT PORT_B,AL      ;посылаем байт в порт B
;---установка регистров ввода/вывода
COM_REG EQU 43H             ;адрес командного регистра
CHAN_2   EQU 42H            ;адрес канала 2
         MOVAL,10110110B    ;цепочка битов для канала 2
         OUT COM_REG,AL     ;засылка в командный регистр
;---засылка счетчика в задвижку
         MOVAX,2705         ;счетчик = 1190000/440
         OUT CHAN_2,AL      ;посылаем младший байт
         MOVAL,AH           ;сдвигаем младший байт в AL
         OUT CHAN_2,AL      ;посылаем старший байт
;---ждем нажатия клавиши
         MOVAH,1            ;номер функции прерывания 21H
         INT  21H           ;вызываем прерывание
;---выключение звука
         IN   AL,PORT_B     ;получаем байт из порта B
         AND AL,11111100B   ;сбрасываем два младших бита
         OUT PORT_B,AL      ;посылаем байт обратно
```

2.2.4. Генерация однотонного сигнала

Некоторым программам требуется набор предостерегающих гудков. Их легко создавать на Бейсике, но операционная система не обеспечивает функцию гудка как таковую и только косвенно позволяет получать доступ к гудку, который вы слышите при старте системы. Для изменения тона вся процедура генерации

звука должна быть запрограммирована на низком уровне. Для того чтобы гудок соответствовал подаваемому им сигналу, необходимо проявить воображение. Для предсказания близкой опасности создайте набор понижающихся тонов [2.2.7] или, если принтер включен, чередуйте гудки динамика компьютера и принтера (вывод кода ASCII 7 на принтер).

Высокий уровень

В Бейсике просто напишите BEEP. Вот кусочек кода, который реагирует на вероятную ошибку гудком и запросом:

```
100 INPUT "Enter your age",AGE          'запрос возраста
110 IF AGE > 100 THEN BEEP:PRINT"Are you really over 100?"
```

Для гудков другой частоты и продолжительности используйте оператор SOUND. Его форма: SOUND частота, длительность , где частота представлена в герцах (3000 - середина диапазона), а длительность - в восемнадцатых долях секунды. SOUND 3000,18 дает гудок длительностью около одной секунды. В приведенном примере динамик быстро переходит от высокого тона к низкому и обратно, "распугивая" все живое в ближайшей окрестности.

```
100 FOR N = 1 TO 200                    'установка числа повторений
110 SOUND 500,1                         'звук низкой частоты на 1 секунду
120 SOUND 5000,1                        'звук высокой частоты на 1 секунду
130 NEXT                                'повтор
```

Средний уровень

Операционная система не предоставляет специальной функции для генерации звука. Но вы можете вызвать знакомый гудок, просто подавая код ASCII 7 на стандартное устройство вывода (терминал), используя одну из функций DOS или BIOS. Код ASCII 7 интерпретируется как управляющий символ "звонок" и не рисуется на экране. Проще всего применять функцию 2 прерывания 21H:

```
MOV  AH,2          ;функция вывода символа на экран
MOV  DL,7          ;посылаем код ASCII 7
INT  21H           ;динамик гудит
```

Низкий уровень

Для простого гудка лучше всего подходит метод, основанный на использовании микросхемы интерфейса с периферией 8255 (см. [1.1.1]). Ниже приведен пример, практически повторяющий гудок, который вы слышите при старте системы.

```
;---гудок динамика
                MOV  DX,800       ;счетчик числа циклов
                IN   AL,61H       ;читаем порт B 8255
                AND  AL,0FEH      ;выключаем бит таймера 8253
NEXTCYCLE:      OR   AL,2         ;включаем бит динамика
                OUT  61H,AL       ;посылаем байт в порт B
                MOV  CX,150       ;длительность первой половины
CYCLEUP:        LOOP CYCLEUP      ;задержка, пока сигнал высокий
                AND  AL,0FDH      ;выключаем бит динамика
                OUT  61H,AL       ;посылаем байт в порт B
CYCLEDOWN:      LOOP CYCLEDOWN    ;задержка, пока сигнал низкий
                DEC  DX           ;уменьшаем счетчик циклов
                JNZ  NEXTCYCLE    ;повторяем цикл, пока DX не 0
```

2.2.5. Генерация набора тонов

В этом параграфе показано, как генерировать цепочку звуков, когда компьютер ничем другим не занят; а в следующем мы рассмотрим, как выполнить ту же задачу, если компьютер занят другой работой. Когда компьютер ничем другим не занят, можно выводить мелодию или производить специальные звуковые эффекты; когда же компьютер занят другой работой, производить звуковые эффекты нельзя.

Создание звуковых строк - одна из мощнейших возможностей, предоставляемых Бейсиком. Построение же строк звуков в ассемблере требует большой работы. Может быть использован любой из двух методов генерации звука, предложенных в [2.2.2] и [2.2.3]. Для обоих методов надо просто генерировать один тон в течении заданного времени, затем следующий и т.д. Каждая звуковая строка формируется из двух строк данных, одна из которых содержит частоты последовательных тонов, а другая хранит их длительности (при условии, что требуются разные длительности). Продолжительность звучания определяется с помощью счетчика времени суток BIOS [2.1.6].

Высокий уровень

Оператор Бейсика PLAY предоставляет большие возможности. Оператор сопровождается строкой нот, перемешанных с информацией о том, как эти ноты должны быть исполнены. Ноты записываются буквами A - G с последующими знаками для диезов и бемолей. Диезы обозначаются знаками # или +, а бемоли - минусом (-). Операторы PLAY "CC#D" и PLAY "CD-D" эквивалентны, но нельзя использовать знаки диезов и бемолей для обозначения звуков, соответствующих белым клавишам. Второй способ задания нот состоит в вычислении кодового номера от 0 до 84, причем 0 соответствует отсутствию звучания, а числа от 1 до 84 соответствуют 84 возможным нотам семи октав, начиная снизу. Номеру должна предшествовать буква N: PLAY "N3N72N44".

Допустимый диапазон - семь октав, внутри каждой могут быть ноты от C(до) до B(си). Октавы пронумерованы от 0 до 6, и нота *до* первой октавы соответствует октаве 3. Текущая октава может быть изменена в любой момент за счет вставки в строку буквы O, за которой следует номер октавы. Если не было начальной установки, то используется октава 4. Оператор PLAY "O3CO4CO5CO6C" выводит ноты C(до) последовательных октав вверх. Другой способ изменения октавы состоит во включении в строку символов > или <, которые переключают тон вверх и вниз на октаву соответственно. Оператор PLAY "O3C>C>C>C" приводит к тому же результату, что и предыдущий.

Длительность нот также может быть изменена за счет вставки кодового номера, которому предшествует буква L. Все последующие ноты будут исполняться с этой длительностью до тех пор, пока не встретится другой код длины. Код - это число от 1 до 64, причем 1 соответствует целой ноте, а 64 - 1/64. Запись L4 соответствует четверти. Темп исполнения нот регулируется кодом темпа, состоящим из буквы T, за которой следует число от 32 до 255, дающее количество четвертей, исполняемых в минуту. Если эти параметры не указаны, то по умолчанию берется длительность L4 и темп 120. Для изменения длительности только одной ноты надо поместить значение длины после ноты и без буквы L. Оператор PLAY "L4CDE16FG" исполнит E как шестнадцатую, а все остальные ноты - как четверти. Длительность пауз берется такой же, как и длительность нот. Поместите число от 1 до 64 после буквы P для паузы. P1 делает паузу длительностью в целую, а P64 - в 1/64. Помещение точки после ноты имеет тот же смысл, что и в обычной музыкальной нотации: длительность ноты увеличивается наполовину. Вторая точка продолжит длительность еще наполовину.

По умолчанию ноты имеют длительность 7/8. Чтобы они звучали с полной длительностью (легато), поместите в строку ML. Если необходимо звучание с длительностью 3/4 (стаккато), поместите в строку MS. Чтобы вернуться к нормальному стилю, надо указать MN.

Обычно вся прочая деятельность программы прекращается до тех пор, пока не будет сыграна строка. Для того чтобы выполнялись операторы, следующие за оператором PLAY, а строка исполнялась в фоновом режиме, поместите в строку MB. Для восстановления нормальной ситуации напишите MF.

Наконец, оператор PLAY позволяет исполнять подстроки внутри длинной строки. Имеется в виду, что часть исполняемой строки может быть введена как обычная строковая переменная, а затем эта переменная может быть вызвана из строки, сформированной в операторе PLAY. Например, если S$ = "EEEEE", то в операторе PLAY "CDXS$;FG" нота E будет повторена 5 раз. Отметим, что имени переменной должна предшествовать буква X, а за именем следовать точка с запятой (;). (Для компилируемых программ применяется другой метод с использованием переменной VARPTR$, детали см. в руководстве по Бейсику.)

В приведенном примере исполняется знакомый бой "дедушкиных" часов. В строке сначала устанавливается стиль исполнения легато, затем темп и начальная октава, и, наконец, четыре ноты, пауза, и те же четыре ноты, но в обратном порядке. Пробелы в строке включены только для удобства программиста - Бейсик игнорирует их.

```
100 PLAY "ML T40 O3 ECD<G P32 G>DEC"
```

Благодаря наличию генератора звука PCjr добавляет к оператору PLAY две возможности. Во-первых, допускается параметр V, устанавливающий громкость. Выражение V5 устанавливает (или изменяет) громкость на уровень 5. Допустимый диапазон от 0 до 15, причем по умолчанию берется 8. 0 полностью подавляет звук. Во-вторых, с помощью оператора PLAY можно одновременно исполнять три звуковые строки. Поместите все три строки в одну программную строку, разделяя их запятыми. Для того чтобы иметь возможность использовать эти специальные свойства, вы должны предварительно разрешить внешний динамик с помощью оператора SOUND ON.

```
100 SOUND ON
110 PLAY ".....","....."."....."
```

Низкий уровень

В примере для генерации звука используется микросхема таймера 8253. Здесь просто исполняются 8 нот, но небольшая модификация может сильно расширить возможности этой процедуры. Имеются три строки данных. Первая устанавливает длительность каждой ноты как кратное произвольного периода задержки (изменяя этот период задержки, можно изменять темп). Вторая строка содержит частоты каждой из 8 нот; эти значения должны быть помещены в регистр задвижки канала 2 микросхемы 8253 для исполнения желаемых тонов. Третья строка содержит мелодию в виде кодовых номеров от 1 до 8, которые соответствуют восьми частотам. Эта строка завершается кодом 0FFH, который служит признаком конца мелодии. Процедура просто читает очередную ноту мелодии, находит соответствующую частоту и помещает ее в канал 2. Затем длительность для этой ноты помещается в счетчик цикла задержки, который использует счетчик времени суток. Когда задержка кончается, переходим к обработке следующей ноты. На рис. 2.5 показана работа этой процедуры.

```
;---в сегменте данных
BEAT     DB  10,9,8,7,6,5,4,3,2   ;длительность нот
FREQ     DW  2280,2031,1809,1709  ;таблица частот
         DW  1521,1353,1207,1139
MELODY   DB  1,2,3,4,5,6,7,8,0FFH ;номер частоты ноты
;---инициализация
PORT_B   EQU 61H
COM_REG  EQU 43H
LATCH2   EQU 42H
         IN  AL,PORT_B           ;получаем текущий статус
         OR  AL,00000011B        ;разрешаем динамик и таймер
         OUT PORT_B,AL           ;заменяем байт
         MOV SI,0                ;инициализируем указатель
         MOV AL,0B6H             ;установка для канала 2
         OUT COM_REG,AL          ;посылаем в командный регистр
;---смотрим ноту, получаем ее частоту и помещаем в канал 2
NEXT:    LEA BX,MELODY           ;берем смещение для мелодии
         MOV AL,[BX][SI]         ;берем код n-й ноты строки
         CMP AL,0FFH             ;проверка на конец строки
         JE  NO_MORE             ;если конец, то на выход
         CBW                     ;переводим в слово
```

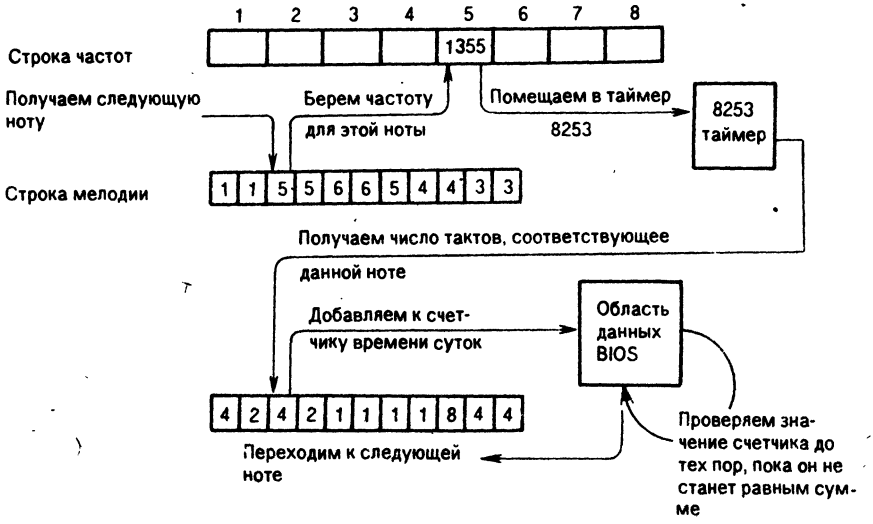


Рис. 2.5. Исполнение строки нот

```
;получение частоты
        MOVBX,OFFSET FREQ  ;смещение таблицы частот
        DEC AX             ;начинаем отсчет с 0
        SHL AX,1           ;умножаем на 2, так как словная таблица
        MOVDI,AX           ;адресуем через DI
        MOVDX,[BX][DI]     ;получаем частоту из таблицы
;начинаем исполнение ноты
        MOVAL,DL           ;готовим младший байт частоты
        OUT LATCH2,AL      ;посылаем его
        MOVAL,DH           ;готовим старший байт частоты
        OUT LATCH2,AL      ;посылаем его
;---создание цикла задержки
        MOVAH,0            ;номер функции чтения счетчика
        INT 1AH            ;получаем значение счетчика
        MOVBX,OFFSET BEAT  ;смещение таблицы длин
        MOVCL,[BX][SI]     ;берем длину очередной ноты
        MOVCH,0
        MOVBX,DX           ;берем младшее слово счетчика
        ADD BX,CX          ;определяем момент окончания
ST_SOU: INT 1AH            ;берем значение счетчика
```

```
        CMP DX,BX           ;сравниваем с окончанием
        JNE ST_SOU          ;неравны - продолжаем звук
        INC SI              ;переходим к следующей ноте
        JMP NEXT
;---завершение
NO_MOR: IN  AL,PORT_B       ;получаем статус порта B
        AND AL,0FCH         ;выключаем динамик
        OUT 61H,AL          ;заменяем байт
```

2.2.6. Генерация строки тонов одновременно с другими операциями

Хотя в Бейсике это делается очень просто, на самом деле подобная задача представляет собой нетривиальный трюк программирования в реальном времени. Для решения такой задачи нужна генерация звука через микросхему 8253 [2.2.3], так как метод, использующий микросхему 8255 [2.2.2], занимает процессор. Соответственно только строки чистых музыкальных тонов могут производиться таким методом - всякого рода звуковые эффекты при этом недоступны. Основная техника программирования в реальном времени показана в [2.1.7]. Программы, работающие в реальном времени, модифицируют прерывание таймера, которое останавливает процессор 18,2 раза в секунду, чтобы изменить показание счетчика времени суток. Расширение процедуры прерывания сравнивает новое значение счетчика времени суток со значением, показывающим время завершения генерации тона, и когда это значение достигнуто, прерывает звук, начинает генерацию другого тона и устанавливает время его окончания.

Высокий уровень

Генерация строки звуков одновременно с другими операциями является одной из возможностей очень мощного оператора PLAY, который детально обсуждался в [2.2.5]. Надо просто добавить в начало управляющей строки MB. Это сокращение от Music Background (фоновая музыка); для того чтобы заставить PLAY прекратить все другие операции, пока генерация звуковой строки не будет завершена, вставьте MF. В приведенном ниже примере во время рисования и заполнения рамки исполняется гамма (для его демонстрации требуется наличие графических возможностей).

```
100 PLAY "MB T100 O3 L4;CDEFG>ABC" 'исполняем набор нот
110 LINE (10,10)-(80,80),1,BF      'одновременно рисуем рамку
```

Низкий уровень

Приведенная процедура является развитием процедуры, показанной в предыдущем параграфе, на случай реального времени. Она требует понимания, как перепрограммировать прерывание таймера, что обсуждалось в [2.1.7]. На эту процедуру должен указывать вектор прерывания и тогда она будет выполняться 18,2 раза в секунду в те моменты, когда будет обновляться значение счетчика времени суток BIOS. Обычно выполняется только несколько строчек, которых достаточно, чтобы определить, что время изменения звука еще не наступило и процедура освободит процессор для решения других задач.

Счетчик времени суток BIOS применяется для измерения длительности каждой ноты. При переходе от одной ноты к другой длительность новой ноты вычисляется как число импульсов счетчика и это значение добавляется к текущему его значению. Каждый раз при вызове процедуры проверяется текущее значение счетчика времени суток, и когда ожидаемое время наконец наступает, выполняется набор операций по поиску новой ноты, программированию ее частоты в канале 2 микросхемы 8253 и установлению нового счетчика длительности. Добавочный код требуется для обработки специальных случаев первой и последней нот в строке.

```
;---в сегменте данных
BEAT      DB 10,9,8,7,6,5,4,3,2     ;длительность нот
FREQ      DW 2280;2031,1809,1709 ;таблица частот
          DW 1521,1355,1207,1139
MELODY    DB 1,2,3,4,5,6,7,8,0FFH ;номер частоты в таблице
HOLDIP    DW 0                      ;запоминаем оригинальный
HOLDCS    DW 0                      ;вектор прерывания
SOU_NOW? DB 1                       ;звук включен?
FIR_NOTE? DB 1                      ;первая нота?
END_NOTE DW 0                       ;счетчик конца ноты
WHICH_NO DW 0                       ;указатель на текущую ноту
;---инициализация вектора прерывания
;изменение вектора
          PUSHDS                    ;сохраняем регистр
          MOV AX,SEG MEL2           ;сегмент процедуры
          MOV DS,AX                 ;помещаем в DS
          MOV DX,OFFSET MEL2        ;смещение процедуры
          MOV AL,1CH                ;номер вектора прерывания
```

```
        MOV AH,25H              ;функция установки вектора
        INT 21H                 ;изменение вектора
        POP DS                  ;восстановление регистра
;
;---программа работает дальше, постоянно вызывая процедуру
;
;---в конце программы восстанавливаем вектор прерывания
        MOV DX,0FF53H           ;восстанавливаем оригинальные
        MOV AX,0F000H           ;значения для вектора 1CH
        MOV DS,AX
        MOV AL,1CH              ;номер прерывания
        MOV AH,25H              ;функция установки вектора
        INT 21H                 ;восстанавливаем вектор
        RET
;---это само прерывание
MEL2    PROC FAR
        PUSH AX                 ;сохраняем изменяемые регистры
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH DI
        PUSH SI
        PUSH DS
        MOV  AX,SS:[114]        ;берем начальный DS со стека
        MOV  DS,AX              ;восстанавливаем его
        CMP  SOU_NOW?,1         ;нужен ли звук?
        JE   PLAY_IT            ;если нет, то выход из прерывания
        JMP  NOT_NOW
PLAY_IT: CMP  FIR_NOTE?,0       ;это первая нота?
        JE   TIM_CHK            ;если нет, то на установку времени
;---инициализация
PORT_B  EQU 61H                 ;определяем имена портов
COM_REG EQU 43H
LATCH2  EQU 42H
        IN   AL,PORT_B          ;берем статус порта B
        OR   AL,00000011B       ;разрешаем динамик и таймер
        OUT  PORT_B,AL          ;посылаем байт обратно
        MOV SI,0                ;указатель на строки
        MOV AL,0B6H             ;инициализация канала 2 таймера
```

```
            OUT  COM_REG,AL          ;посылаем в командный регистр
            MOV  FIR_NOTE?,0         ;сбрасываем флаг первой ноты
;---ищем ноту, получаем ее частоту, посылаем в канал 2
NEXT:       LEA  BX,MELODY           ;берем смещение строки мелодии
            MOV  SI,WHICH_NO         ;указатель на текущую ноту
            MOV  AL,[BX][SI]         ;код текущей ноты строки
            CMP  AL,0FFH             ;проверяем признак конца
            JE   NO_MOR              ;если да, то на конец
            CBW                      ;иначе - в словный формат
;получаем частоту
            MOV  BX,OFFSET FREQ      ;смещение таблицы частот
            DEC  AX                  ;начинаем отсчет с нуля
            SHL  AX,1                ;умножаем на 2, так как словная таблица
            MOV  DI,AX               ;адресуемся через DI
            MOV  DX,[BX][DI]         ;получаем частоту из таблицы
;начинаем исполнение ноты
            MOV  AL,DL               ;готовим младший байт частоты
            OUT  LATCH2,AL           ;посылаем в регистр задвижки
            MOV  AL,DH               ;готовим старший байт
            OUT  LATCH2,AL           ;посылаем его
;---пустой цикл, определяющий длительность нот
TIME_IT:    MOV  AH,0                ;функция чтения счетчика
            INT  1AH                 ;получаем значение счетчика
            MOV  BX,OFFSET BEAT      ;смещение строки длин нот
            MOV  CL,[BX][SI]         ;длительность текущей ноты
            MOV  CH,0
            MOV  BX,DX               ;младшее слово значения счетчика
            ADD  BX,CX               ;добавляем длину в импульсах
            MOV  END_NOTE,BX         ;запоминаем время окончания
TIM_CHK:    MOV  AH,0                ;функция чтения счетчика
            INT  1AH                 ;читаем счетчик
            CMP  DX,END_NOTE         ;сравниваем с нужным
            JNE  NO_NOW              ;если неравно, то выходим
            MOV  SI,WHICH_NO         ;иначе, берем следующую ноту
            INC  SI                  ;увеличиваем номер ноты
            MOV  WHICH_NO,SI         ;запоминаем его
            JMP  NEXT_NOTE           ;начинаем следующую ноту
;---завершение процедуры
NO_MOR:     IN   AL,PORT_B           ;берем статус порта B
```

```
            AND AL,0FCH             ;выключаем динамик
            OUT 61H,AL              ;возвращаем байт
            MOV SOU_NOW?,0          ;восстанавливаем переменные
            MOV FIR_NOTE?,1
NO_NOW:     POP DS                  ;восстанавливаем регистры
            POP SI
            POP DI
            POP DX
            POP CX
            POP BX
            POP AX
            IRET                    ;возврат из прерывания
MELODY2 ENDP
```

2.2.7. Создание плавного перехода тонов

Плавные переходы тонов производятся за счет непрерывного изменения частоты. Этого можно достигнуть как в Бейсике, так и программируя на ассемблере. Такой звуковой эффект можно сделать более выразительным, если немного уменьшать длительность каждого сегмента тона при повышении звука или слегка увеличивать длительность при понижении.

Высокий уровень

В Бейсике надо просто поместить оператор SOUND [2.2.2] в цикл, используя очень малые длины тонов. При каждом новом проходе цикла надо увеличивать частоту. В [2.2.8] приведен пример использования оператора PLAY для более быстрых переходов.

```
100 FOR N = 1 TO 500 STEP 15
110 SOUND 400 + N,1
120 NEXT
```

Низкий уровень

Проще всего применять метод генерации звука, управляемый микросхемой интерфейса с периферией 8255. Меняйте значение бита 1 порта B между 0 и 1, используя для отсчета времени пустой цикл, как показано в [2.2.2]. При начале каждого нового

пустого цикла, за счет засылки значения в CX, слегка изменяйте это значение. Здесь тон повышается:

```
;---запрет микросхемы таймера
PB    EQU  61H          ;адрес порта B микросхемы 8255
      IN   AL,PB        ;получаем из него байт
      OR   AL,1         ;сбрасываем бит 0
      OUT  PB,AL        ;возвращаем байт в порт
;---установка частоты и длительности звука
      MOV  BX,9000      ;начальное значение счетчика
      MOV  DX,3000      ;длительность звука 3000 циклов
REPEAT:                 ;сюда возвращаемся после цикла
;---установка бита динамика
      OR   AL,00000010B ;устанавливаем бит 1
      OUT  PB,AL        ;посылаем байт в порт B
      MOV  CX,BX        ;установка счетчика для 1/2 цикла
CYC1: LOOP CYC1         ;пустой цикл на 1000 повторов
;---сброс бита динамика
      AND  AL,11111101B ;сбрасываем бит 1
      OUT  PB,AL        ;посылаем байт в порт
      MOV  CX,BX        ;установка счетчика
CYC2: LOOP CYC2         ;пустой цикл
;---переход к следующему циклу
      DEC  BX           ;увеличиваем частоту, уменьшая
      DEC  BX           ;счетчик
      DEC  DX           ;уменьшаем оставшуюся длительность
      JNZ  REPEAT       ;если DX не 0, то новый цикл
```

Этот простой метод приводит к тому, что высокие тона проходят значительно быстрее, чем низкие. Для коротких интервалов такой эффект может быть желательным, а когда он не нужен, надо добавить код, который при повышении тона пересылает в DX большие значения на следующем цикле.

2.2.8. Создание звуковых эффектов

Звуковые эффекты обычно достигаются непрерывным изменением частоты тона. Только PCjr достаточно хорошо оборудован для этой цели (см. [2.2.1]). На остальных машинах нельзя производить звуковые эффекты одновременно с другими операциями.

Высокий уровень

Благодаря мощности операторов SOUND и PLAY Бейсик позволяет достаточно легко создавать сложные звуковые эффекты. Но все должно быть сконструировано из чистых музыкальных тонов, а это значит, что эффект дисторции звука должен достигаться за счет такого быстрого изменения тона, что ухо не успевает разделить тона. Например, душераздирающее "чириканье" может быть получено при быстром переключении между одним и тем же тоном, отстоящим на несколько октав:

```
100 FOR N = 1 TO 100          'установка длительности
110 PLAY "L64 T255"           'самый быстрый темп
120 PLAY "O1A"                'выдаем низкое А
130 PLAY "O5A"                'выдаем высокое А
140 NEXT                      'повтор
```

При изменении частоты всего на несколько герц получаем вибрацию:

```
100 FOR N = 1 TO 100          'установка длительности
110 SOUND 440,1               'выдаем ноту А
120 SOUND 445,1               'немного меняем частоту
130 NEXT                      'повтор
```

Другой метод заключается во вложении плавно меняющихся тонов внутрь последовательности, которая сама "гуляет" по частотам вверх или вниз. На рис. 2.6. показана движущаяся вверх последовательность. Этот метод применяется во многих играх с лабиринтами.

```
100 FOR I = 1 TO 10           'число повторений
110 FOR J = 1 TO 6            'число разных октав
120 PLAY "mbl64t255O = j;ba #ag #gf #fed #dc #cc #dd #eff #gg #aa #b"
130 NEXT                      'повтор в более высокой октаве
140 NEXT                      'повтор всей последовательности
```

PCjr значительно более мощный, чем остальные машины, благодаря специальной микросхеме генератора звука. Оператор NOISE может производить множество звуков, формат этого оператора такой:

NOISE источник, громкость, длительность

Источник - это число от 0 до 7, значение которого приведено в таблице:

0	периодический шум в высоком диапазоне
1	периодический шум в среднем диапазоне
2	периодический шум в низком диапазоне
3	периодический шум, диапазон меняется с каналом 3
4	белый шум в высоком диапазоне
5	белый шум в среднем диапазоне
6	белый шум в низком диапазоне
7	белый шум,диапазон меняется с каналом 3

Громкость задается числом от 0 до 15, где 0 соответствует отсутствию звука. Длительность указывается числом импульсов счетчика времени суток, которые отсчитываются 18,2 раза в секунду.

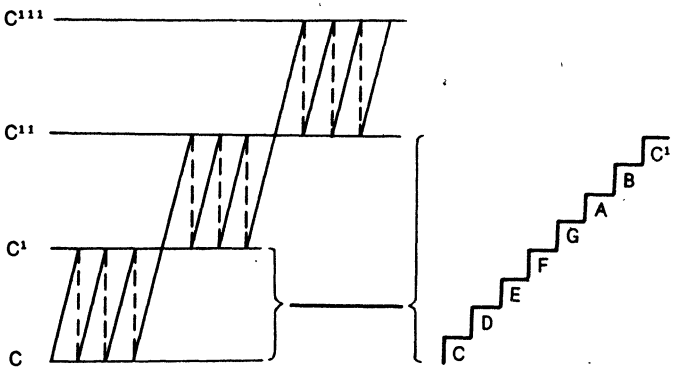


Рис. 2.6. Звуковой эффект скользящего тона

Низкий уровень

Любой из способов, показанных на Бейсике, может быть реализован на ассемблере, хотя, как видно из предыдущих параграфов, это требует затрат на программирование. Кроме того,

ассемблер позволяет генерировать нечистые тона, когда интервал, в течение которого динамик включен, не равен интервалу, в течение которого он выключен. Такое нарушение симметрии может приводить к жужжащим и брякающим звукам. Когда отношение этих интервалов составляет, скажем, 50 к 1, получаем жужжание. Если увеличить отношение еще в 10 - 20 раз, то жужжание переходит в отдельные брякающие звуки. В любом случае звук генерируется микросхемой интерфейса с периферией 8255 с помощью методов, показанных в [2.2.2]. Вот пример жужжания:

```
NUM_CES     EQU  300          ;число переключений динамика
FREQ1       EQU  50           ;время, когда динамик включен
FREQ2       EQU  3200         ;время, когда динамик выключен
PORT_B      EQU  61H          ;адрес порта B микросхемы 8255
            CLI               ;запрет прерываний
            MOV  DX,NUM_CES   ;DX считает длину тона
            IN   AL,PORT_B    ;получаем статус порта
            AND  AL,11111110B ;отключаем динамик от таймера
NEXT_CLE:   OR   AL,00000010B ;включаем динамик
            OUT  PORT_B,AL    ;посылаем команду
            MOV  CX,FREQ1     ;задержка для первой части
FIRST_HALF: LOOP FIRST_HALF
            AND  AL,11111101B ;выключаем динамик
            OUT  PORT_B,AL    ;посылаем команду
            MOV  CX,FREQ2     ;задержка для второй части
SEC_HALF:   LOOP SECND_HALF
            DEC  DX           ;уменьшаем число циклов
            JNZ  NEXT_CLE     ;если 0, то пора заканчивать
            STI               ;разрешаем прерывания
```

Для создания брякающих звуков можно использовать этот же код, но надо заменить значение FREQUENCY2 на величину около 40000.

2.2.9. Одновременная генерация разных звуков

Только микросхема генератора звука, имеющаяся в PCjr, способна одновременно генерировать разные звуки (см. [2.2.1]). Однако ассемблер позволяет объединить два способа генерации звука, что создает имитацию одновременной генерации двух

разных звуков. Интерференция этих двух сигналов приводит к сложной форме звуковой волны. Каждый из двух звуков имеет меньшую громкость, поэтому в результате получается скорее жужжание, чем два разных голоса. Этот прием реально полезен только для создания звуковых эффектов.

Низкий уровень

Надо просто объединить два метода генерации звука, показанные в [2.2.2] и [2.2.3]. Начните звук через канал 2 микросхемы таймера. Затем модулируйте выход динамика за счет бита 1 порта B микросхемы интерфейса с периферией. Второе действие определяет продолжительность звука. Не забудьте выключить микросхему таймера при завершении.

```
;---начинаем генерацию звука через канал 2 таймера
            IN    AL,61H          ;получаем байт из порта B
            OR    AL,3            ;устанавливаем младшие два байта
            OUT   61H,AL          ;посылаем байт обратно
            MOV   AL,10110110B    ;цепочка для командного регистра 8253
            OUT   43H,AL          ;посылаем в регистр
            MOV   AX,600H         ;счетчик для канала 2
            OUT   42H,AL          ;посылаем младший байт
            MOV   AL,AH           ;готовим старший байт
            OUT   42H,AL          ;посылаем старший байт
;---генерируем вторую частоту микросхемой 8255
NUM_CES     EQU   9000            ;число переключений
FREQ        EQU   150             ;задержка для половины цикла
            CLI                   ;запрет прерываний
            MOV   DX,NUM_CES      ;DX считает длину тона
            IN    AL,61H          ;получаем статус порта
            AND   AL,11111111B    ;отключаем динамик от таймера
NEX_CLE:    OR    AL,00000010B    ;включаем динамик
            OUT   61H,AL          ;посылаем назад в порт
            MOV   CX,FREQ         ;задержка на 1/2 цикла
FIR_HAF:    LOOP  FIR_HAF
            AND   AL,11111101B    ;выключаем динамик
            OUT   61H,AL          ;посылаем команду в порт
            MOV   CX,FREQ         ;задержка на 1/2 цикла
SEC_HAF:    LOOP  SEC_HAF
            DEC   DX              ;меняем счетчик циклов
```

```
        JNZ   NEX_CLE          ;если 0, то пора заканчивать
        STI                    ;разрешаем прерывания
;---выключение канала 2 микросхемы таймера
        IN    AL,61H           ;получаем статус порта
        AND   AL,11111100B     ;сбрасываем 2 младших бита
        OUT   61H,AL           ;посылаем байт обратно
```

Глава 3. Клавиатура

3.1. Управление клавиатурой

Клавиатура содержит интелевский микропроцессор, который воспринимает каждое нажатие на клавишу и выдает скан-код в порт А микросхемы интерфейса с периферией [1.1.1], расположенной на системной плате. Скан-код - это однобайтовое число, младшие 7 бит которого представляют идентификационный номер, присвоенный каждой клавише. Таблица скан-кодов приведена в [3.3.2]. На всех машинах, кроме AT, старший бит кода говорит о том, была ли клавиша нажата (бит = 1, код нажатия) или освобождена (бит = 0, код освобождения). Например, 7-битовый скан-код клавиши B - 48, или 110000 в двоичной системе. Когда эту клавишу нажимают, в порт А посылается код 10110000, а когда отпускают - код 00110000. Таким образом, каждое нажатие на клавишу дважды регистрируется в микросхеме 8255. И всякий раз микросхема 8255 выдает подтверждение микропроцессору клавиатуры. AT работает немного по-другому, посылая в обоих случаях один и тот же скан-код, но предваряя его кодом F0H, если клавиша отпускается.

Когда скан-код выдается в порт А, вызывается прерывание клавиатуры (INT 9). Процессор моментально прекращает свою работу и выполняет процедуру, анализирующую скан-код. При поступлении кода от клавиши сдвига или переключателя изменение статуса записывается в память. Во всех остальных случаях скан-код трансформируется в код символа при условии, что он подается при нажатии клавиши (в противном случае скан-код отбрасывается). Конечно, процедура сначала определяет установку клавиш сдвига и переключателей, чтобы правильно

получить вводимый код (это "a" или "A"?). Затем введенный код помещается в буфер клавиатуры, который является областью памяти, способной запомнить до 15 вводимых символов, пока программа слишком занята, чтобы обработать их. На рис. 3.1 показан путь, который проходит нажатие на клавишу перед тем, как попасть в вашу программу.

Имеются два типа кодов символов: коды ASCII и расширенные коды. Коды ASCII - это байтовые числа, соответствующие расширенному набору кодов ASCII для IBM PC, приведенному в [3.3.3]. Для IBM PC этот набор включает обычные символы пишущей машинки, а также ряд специальных букв и символов псевдографики. Коды ASCII включают также 32 управляющих кода, которые обычно используются для передачи команд периферийным устройствам, а не выводятся как символы на экране; однако каждый из них имеет соответствующий символ, который может быть выведен на дисплей, с использованием прямой адресации дисплейной памяти [4.3.1]. (Строго говоря, только первые 128 символов являются настоящими символами ASCII, так как ASCII - это аббревиатура, означающая American Standard Code for Information Interchange (американский стандартный код для обмена информацией). Но программисты обычно говорят о кодах ASCII, чтобы отличить их от других чисел. Например, ASCII 8 относится к клавише «Backspace», в то время как 8 - это цифра, которой соответствует ASCII 56&.)

Второй набор кодов, расширенные коды; присвоен клавишам или комбинациям клавиш, которые не имеют представляющего их символа ASCII, таким, как функциональные клавиши или комбинации с клавишей Alt. Расширенные коды имеют длину 2 байта, причем первый байт всегда ASCII 0. Второй байт - номер расширенного кода (список кодов приведен в [3.3.5]). Например, код 0:30 представляет Alt-A. Начальный ноль позволяет программе определить, принадлежит ли данный код набору ASCII или расширенному набору.

Существует несколько комбинаций клавиш для выполнения специальных функций, они не генерируют скан-коды. Эти комбинации включают «Ctrl-Break», «Ctrl-Alt-Del» и «PrtSc» плюс «SysReq» для AT и «Ctrl-Alt-стрелка влево, -стрелка вправо, -CapsLock, -Ins» для PCjr. Эти исключения приводят к заранее предопределенным результатам [3.3.2]. Все остальные нажатия клавиш должны интерпретироваться вашей программой, и если они имеют специальное назначение, скажем сдвинуть курсор влево, то ваша программа должна содержать код для достижения этого эффекта.

К счастью, операционная система позволяет выполнять различные процедуры для чтения кодов из буфера клавиатуры, включая

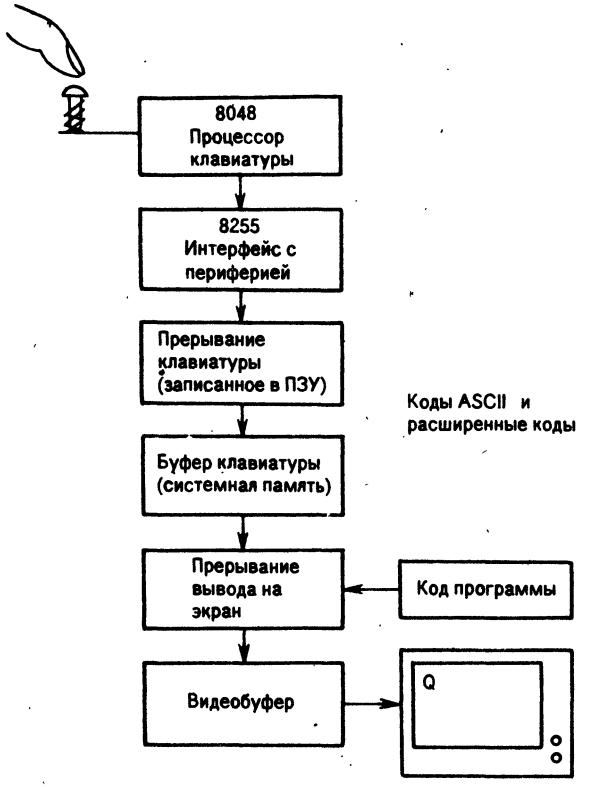


Рис. 3.1. От клавиатуры к экрану

средства для получения сразу целой строки. Поскольку эти процедуры обеспечивают практически все, что вы можете пожелать, бессмысленно писать свои программы обработки ввода с клавиатуры, и поэтому в данной главе очень мало примеров программирования на низком уровне. В ней обсуждаются вопросы, связанные с перепрограммированием прерывания клавиатуры.

3.1.1. Очистка буфера клавиатуры

Программа должна очистить буфер клавиатуры перед тем, как выдать запрос на ввод, исключая тем самым посторонние нажатия клавиш, которые могут к тому времени накопиться в буфере.

Буфер может накапливать до 15 нажатий на клавишу независимо от того, являются они однобайтовыми кодами ASCII или двухбайтовыми расширенными кодами. Таким образом, буфер должен отвести два байта памяти для каждого нажатия на клавишу. Для однобайтовых кодов первый байт содержит код ASCII, а второй - скан-код клавиши. Для расширенных кодов первый байт содержит ASCII 0, а второй - номер расширенного кода. Этот код обычно совпадает со скан-кодом клавиши, но не всегда, поскольку некоторые клавиши можно комбинировать с клавишами сдвига для генерации различных кодов.

Буфер устроен, как циклическая очередь, которую называют также буфером FIFO (первый вошел - первый ушел). Он занимает непрерывную область адресов памяти. Однако определенной ячейки памяти, которая хранит "начало строки" в буфере, нет. Вместо этого два указателя хранят позиции головы и хвоста строки символов, находящейся в буфере в текущий момент. Новые нажатия клавиш сохраняются в позициях, следующих за хвостом (в более старших адресах памяти), и соответственно обновляется указатель хвоста буфера. После того как израсходовано все буферное пространство, новые символы продолжают вставляться, начиная с самого начала буферной области; поэтому возможны ситуации, когда голова строки в буфере имеет больший адрес, чем хвост. Когда буфер заполнен, новые вводимые символы игнорируются, при этом прерывание клавиатуры выдает гудок через динамик. На рис. 3.2 показаны некоторые возможные конфигурации данных в буфере.

В то время как указатель на голову установлен на первый введенный символ, указатель на хвост установлен на позицию за последним введенным символом. Когда оба указателя равны, буфер пуст. Чтобы разрешить ввод 15 символов, требуется 16-я пустая позиция, 2 байта которой всегда содержат код возврата каретки (ASCII 13) и скан-код клавиши «Enter», равный 28. Эта пустая позиция непосредственно предшествует голове строки символов. 32 байта буфера начинаются с адреса 0040:001E. Указатели на голову и хвост расположены по адресам 0040:001A и 0040:001C соответственно. Хотя под указатели отведено 2 байта, используется только младший байт. Значения указателей меняются от 30 до 60, что соответствует позициям в области данных BIOS. Для очистки буфера надо просто установить значение ячейки 0040:001A равным значению ячейки 0040:001C.

Отметим, что программа может вставлять символы в буфер, завершая строку символом возврата каретки и меняя значения указателей. Если это сделано перед завершением программы, то при возврате управления в MS-DOS эти символы будут считаны и может быть автоматически загружена другая программа.

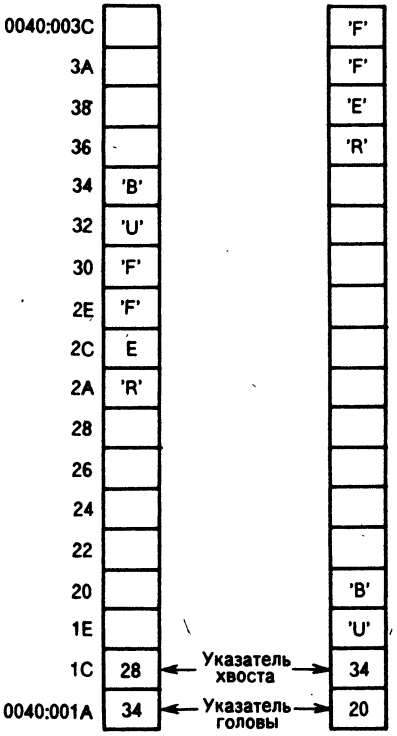


Рис. 3.2. Конфигурация буфера клавиатуры

Высокий уровень

В Бейсике для получения и изменения значений указателей буфера используются операторы PEEK и POKE:

```
100 DEF SEG = &H40                    'устанавливаем значение сегмента
110 POKE &H1C, PEEK(&H1A)             'выравниваем указатели
```

Этот метод не самый лучший. Некоторые программы могут создавать буфер где-нибудь в другом месте памяти, а кроме того, всегда существует возможность, прерывания клавиатуры в середине строки 110, которое изменит указатель хвоста. По этим причинам лучше оставить указатели буфера в покое. Нужно читать из

буфера до тех пор, пока не будет возвращен символ ASCII 0, показывающий, что буфер пуст:

```
100 IF INKEY$<>"" THEN 100          'берем следующее, если не пуль
```

Средний уровень

Функция 0C прерывания 21H выполняет любую из функций ввода с клавиатуры 1, 6, 7, 8 и A (описанных в данной главе), но перед этим чистит буфер клавиатуры. Надо просто поместить номер функции ввода в AL (в этом примере - 1):

```
;---очистка буфера перед ожиданием нажатия клавиши
MOV  AH,0CH          ;выбираем функцию DOS 0CH
MOV  AL,1            ;выбираем функцию ввода символа
INT  21H             ;чистим буфер, ждем ввода
```

Низкий уровень

Как и в примере высокого уровня, приравниваем значение указателя на хвост значению указателя на голову. Для избежания влияния прерывания клавиатуры запрещаем прерывания на время модификации указателя:

```
;---выравниваем значения указателей на голову и хвост
CLI                  ;запрещаем прерывания
SUB  AX,AX           ;обнуляем регистр
MOV  ES,AX           ;добавочный сегмент - с начала памяти
MOV  AL,ES:[41AH]    ;берем указатель на голову буфера
MOV  ES:[41CH],AL    ;посылаем его в указатель хвоста
STI                  ;разрешаем прерывания
```

3.1.2. Проверка символов в буфере

Вы можете проверить, был ли ввод с клавиатуры, не удаляя символ из буфера клавиатуры. Буфер использует два указателя, которые отмечают голову и хвост очереди символов, находящихся в буфере в текущий момент. Когда значения этих указателей равны, буфер пуст. Надо просто сравнить содержимое ячеек памяти 0040:001A и 0040:001C. (Нельзя просто проверить символ, находящийся в голове очереди, поскольку буфер организован в виде циклической очереди и позиция ее головы постоянно меняется [3.1.1].)

Высокий уровень

Надо просто использовать оператор PEEK для получения значений, а затем сравнить их:

```
100 DEF SEG = &H40 'устанавливаем сегмент на начало памяти
110 IF PEEK(&H1A)<>PEEK(&H1C) THEN ... '...то буфер не пуст
```

Средний уровень

Функция 0BH прерывания 21H возвращает значение 0FFH в регистре AL, когда буфер клавиатуры содержит один или более символов, и значение 0, когда буфер пуст:

```
;---проверка наличия символа в буфере
MOV  AH,0BH             ;номер функции
INT  21H                ;вызываем прерывание 21H
CMP  AL,0FFH            ;сравниваем с 0FFH
JE   GET_KEYSTROKE      ;переход, если буфер не пуст
```

Функция 1 прерывания BIOS 16H предоставляет ту же возможность, но, кроме того, она показывает, какой символ в буфере. Флаг нуля (ZF) сбрасывается, если буфер пуст, и устанавливается, если в буфере есть символ. В последнем случае копия символа, находящегося в голове буфера, помещается в AX, но символ из буфера не удаляется. В AL возвращается код символа для однобайтовых символов ASCII, иначе - ASCII 0 для расширенных кодов, и тогда номер кода пересылается в AH.

```
;---проверяем наличие символа в буфере
MOV  AH,1               ;номер функции
INT  16H                ;проверка наличия символа
JZ   NO_CHARACTER       ;переход, если ZF = 1
;---имеется символ - смотрим какой
CMP  AL,0               ;это расширенный код?
JE   EXTENDED_CODE      ;если да, то на другую ветку
```

Низкий уровень

Как и в примере высокого уровня, сравниваем указатели:

```
;---сравниваем указатели на голову и хвост
MOV  AX,0               ;устанавливаем добавочный сегмент
```

```
MOV  ES,AX              ;на начало памяти
MOV  AL,ES:[41AH]       ;берем один указатель
MOV  AH,ES:[41CH]       ;берем другой указатель
CMP  AH,AL              ;сравниваем их
JNE  GET_KEYSTROKE      ;если неравны, то к процедуре ввода
```

3.1.3. Ожидание ввода символа без вывода его на экран

Обычно вводимые символы выводятся на экран, чтобы было видно, что напечатано. Но иногда автоматическое эхо на экране нежелательно. Например, выбор пункта меню по нажатию клавиши. Часто надо сначала проверить вводимые символы на ошибку перед выводом на экран. В частности, любая программа, обрабатывающая расширенные коды, должна избегать автоматического эха, так как при этом первый байт этих кодов (ASCII 0) будет выводиться на экран, вставляя пробелы между символами.

Высокий уровень

Функция Бейсика INKEY$ не дает эха на терминал. Она возвращает строку длиной в 1 байт для символов ASCII и длиной в 2 байта для расширенных кодов. INKEY$ не ожидает нажатия клавиши до тех пор, пока она не помещена в цикл, в котором ожидается нажатие клавиши. Цикл работает, обращаясь к INKEY$, а затем присваивая возвращаемую им строку переменной, в данном случае C$. Если клавиша не была нажата, то INKEY$ возвращает нулевую строку, т.е. строку длиной в ноль символов, которая обозначается двумя знаками кавычек, между которыми ничего нет (""). До тех пор пока INKEY$ возвращает "", цикл повторяется:

```
100 C$ = INKEY$:IF C$ = "" THEN 100
```

В приведенном ниже примере предполагается, что вводимые символы выбирают одну из возможностей меню и каждый выбор приводит к выполнению определенной процедуры программы. Выбор может быть сделан за счет нажатия клавиш A, B, C ... (давая 1-байтовые коды ASCII) или Alt-A, Alt-B, Alt-C ... (давая 2-байтовые расширенные коды). Для их распознавания используется функция LEN, которая определяет, была ли строка длиной в 1 или 2 байта. При кодах ASCII набор операторов IF...THEN сразу начинает проверять, какая клавиша была нажата, отсылая программу на соответствующую процедуру. При 2-байтовых кодах управление передается отдельной процедуре. В этой процедуре функция RIGHT$ убирает левый символ, который просто равен

нулю, и только отмечает расширенный код. Затем используется функция ASC для преобразования строки из символьной формы в числовую. И наконец, вторая серия операторов IF...THEN проверяет получившееся число на соответствие кодам Alt-A, Alt-B и т.д.

```
100 C$ = INKEY$:IF C$ = "" THEN 100 'ожидаем нажатия клавиши
110 IF LEN(C$) = 2 THEN 500         'если расширенный код - на 500
120 IF C$ = "a" OR C$ = "A" THEN GOSUB 1100 'это A?
130 IF C$ = "b" OR C$ = "B" THEN GOSUB 1200 'это B?
140 IF C$ = "c" OR C$ = "C" THEN GOSUB 1300 'это C?
.
.
500 C$ = RIGHT$(C$,1)               'получаем второй байт расширенного кода
510 C = ASC(C$)                     'преобразуем его в число
520 IF C = 30 THEN GOSUB 2100       'это Alt-A?
530 IF C = 48 THEN GOSUB 2200       'Alt-B?
540 IF C = 46 THEN GOSUB 2300       'Alt-C?
```

Отметим, что в строке 120 (и последующих) можно также использовать числовые значения кодов ASCII:

```
120 IF C = 97 OR C = 65 THEN GOSUB 1100
```

Конечно, надо сначала преобразовать C$ в форму целого числа, как это сделано в строке 510. В программах, в которых требуется длинная цепочка таких операторов, можно сэкономить место, изменяя C таким образом, чтобы код буквы всегда соответствовал либо верхнему, либо нижнему регистру. Сначала нужно только проверить, что код ASCII C$ находится в правильном диапазоне. Затем установить, меньше ли этот код 91, тогда мы имеем дело с символом верхнего регистра. Если это так, то надо для перевода в нижний регистр добавить 32. В противном случае оставить все как есть. После этого будет достаточно более короткого оператора, такого, как IF C = 97 THEN ... Вот код этой процедуры:

```
500 C = ASC(C$)                          'получаем код ASCII символа
510 IF NOT ((C>64 AND C<91)OR(C>96 AND C<123)) THEN ...
520 IF C<91 THEN C = C + 32              'приводим все к нижнему регистру
530 IF C = 97 THEN ...                   ... начинаем проверку значений
```

Средний уровень

Функции 7 и 8 прерывания 21H ожидают ввода символа, если буфер клавиатуры пуст; появление символа на экране не отобра-

жается. При этом функция 8 распознает «Ctrl-Break» (и инициирует процедуру обработки «Ctrl-Break» [3.2.8]), а функция 7 не реагирует на него. В обоих случаях символ возвращается в AL. Когда AL содержит ASCII 0, получен расширенный код. Повторите прерывание, и в AL появится второй байт расширенного кода.

```
;---получаем введенный символ
                MOV     AH,7            ;номер функции
                INT     21H             ;ожидаем ввод символа
                CMP     AL,0            ;проверка на расширенный код
                JE      EXT_COD         ;если да, то на особую процедуру
                                        ;иначе - код символа в AL
.
.
;---процедура обработки расширенных кодов
EXT_COD:        INT     21H ;берем второй байт кода
                CMP     AL,75           ;проверяем на "стрелку-влево"
                JNE     C_R             ;если нет, то следующая проверка
                JMP     CUR_LEFT        ;если да, то на процедуру
C_R:            CMP     AL,77           ;сравниваем дальше и т.д.
```

BIOS обеспечивает процедуру, которая предоставляет те же возможности, что и функции MS-DOS. Поместите 0 в AH и вызовите прерывание 16H. Функция ожидает ввода символа и возвращает его в AL. В этом случае и расширенные коды обрабатываются за одно прерывание. Если в AL содержится 0, то в AH будет содержаться номер расширенного кода. При этом не обрабатывается «Ctrl-Break».

```
;---ждем нажатия клавиши
                MOV     AH,0            ;номер функции ожидания ввода
                INT     16H             ;получаем введенный код
                CMP     AL,0            ;проверка на расширенный код
                JE      EXT_COD         ;если да, то на специальную процедуру
                                        ;иначе,символ в AL
;---процедура обработки расширенного кода
EXT_COD:        CMP     AH,75           ;берем расширенный код из AH
                                        ;и т.д.
```

3.1.4. Ожидание нажатия клавиши и вывод эха на экран

При вводе данных и текста эхо вводимых символов обычно выдается на экран. При этом такие символы, как возврат каретки

или забой, переводятся в соответствующие перемещения курсора, а не изображаются как ASCII-символы для этих кодов. Выдача эха происходит в той позиции, где предварительно был установлен курсор, и текст автоматически переносится на следующую строку при достижении конца текущей. Перенос на следующую строку не требует специального кода, поскольку символы помещаются в следующую позицию буферной памяти дисплея, которая представляет собой одну длинную строку, включающую все 25 строк дисплея.

Высокий уровень

В Бейсике надо перехватить введенный символ оператором INKEY$, как показано в [3.1.3]. Затем его надо вывести на экран, прежде чем получать таким же способом следующий. Для вывода можно использовать либо оператор PRINT, либо оператором POKE прямо поместить символ в видеобуфер, с помощью отображения в память, как показано в [4.3.1] (буфер начинается с сегмента памяти &HB000 для монохромного адаптера и с &HB800 для цветного адаптера). При использовании PRINT не забудьте поставить в конце двоеточие, иначе будет автоматически добавлен код возврата каретки. Ниже приведены примеры применения обоих методов. При этом не проводится никакого анализа на управляющие символы. Вводимые символы формируются в виде строки данных в переменной KEYSTROKE$.

```
100 ' метод, использующий PRINT
110 LOCATE 10,40                        'установка курсора в позицию 10,40
120 KEYSTR$ = ""                        'очистка переменной
130 C$ = INKEY$:IF C$ = "" THEN 130     'ожидание ввода символа
140 KEYSTR$ = KEYSTR$ + C$              'запись его в переменную
150 PRINT C$;                           'печать символа
160 GOTO 130                            'прием следующего символа

100 ' метод, использующий POKE
110 DEF SEG = &HB000                    'установка сегмента на видеобуфер
120 POINT = 1678                        'указатель на позицию 10,40
130 KEYSTR$ = ""                        'очистка переменной
140 C$ = INKEY$:IF C$ = "" THEN 140     'ожидание ввода символа
150 KEYSTR$ = KEYSTR$ + C$              'запись его в переменную
160 POKE POINTER,ASC(C$)                'помещение символа в видеобуфер
170 POINT = POINT + 2                   'сдвиг указателя на следующий символ
180 GOTO 140                            'прием следующего символа
```

Средний уровень

Функция 1 прерывания 21H ожидает ввода символа, если буфер клавиатуры пуст, а затем выводит его на экран в текущую позицию курсора. Обрабатывается «Ctrl-Break», поэтому может выполняться процедура обработки «Ctrl-Break» [3.2.8]. Введенный символ возвращается в AL. При вводе расширенного кода AL содержит ASCII 0. Для получения в AL второго байта расширенного кода надо повторить прерывание.

```
;---получение введенного символа
        MOV  AH,1        ;номер функции
        INT  21H         ;ожидаем нажатия клавиши
        CMP  AL,0        ;расширенный код?
        JE   EXT_COD     ;если да, то на специальную процедуру
                         ;иначе - символ находится в AL

;---процедура обработки расширенных кодов
        INT  21H         ;получаем в AL номер кода
        CMP  AL,77       ;проверка на "курсор-вправо"
        JNE  C_R         ;если нет, проверка следующего
        JMP  CUR_RIT     ;если да, то на процедуру
C_R:    CMP  AL,75       ;... и т.д.
```

Эта функция полностью игнорирует клавишу «ESC». Клавиша табуляции интерпретируется нормально. Клавиша «Backspace» сдвигает курсор на одну позицию влево, но символ, находящийся в этой позиции, не стирается. Клавиша «Enter» вызывает перемещение курсора в первую позицию текущей строки (нет автоматического перевода строки).

3.1.5. Прием символа без ожидания

Некоторые программы, работающие в реальном времени, не могут останавливаться и ждать нажатия клавиши; они принимают символ из буфера клавиатуры только в те моменты, когда это удобно для программы. Например, бездействие процессора во время ожидания ввода с клавиатуры остановило бы все действия на экране в игровой программе. Напомним, что с помощью методов, описанных в [3.1.2], легко проверить, пуст или нет буфер клавиатуры.

Высокий уровень

Надо просто использовать INKEY$, не помещая его в цикл:

```
100 C$ = INKEY$                    'получение символа
110 IF C$ <> "" THEN...            'если символ введен, то ...
120 ...                            'иначе - нет символа в буфере
```

Средний уровень

Функция 6 прерывания 21H - это единственный способ получить введенный символ без ожидания. Эта функция не дает эха на экран и не распознает «Ctrl-Break». Перед вызовом прерывания в DL должно быть помещено 0FFH. В другом случае функция 6 служит совершенно противоположной цели - печатает в текущей позиции курсора символ, находящийся в DL. Флаг нуля устанавливается в 1, если буфер клавиатуры пуст. Если символ принят, то он помещается в AL. Код ASCII 0 индицирует расширенный код, и для получения номера кода прерывание должно быть повторено.

```
            MOV   AH,6          ;номер функции DOS
            MOV   DL,0FFH       ;запрос ввода с клавиатуры
            INT   21H           ;получение символа
            JZ    NO_CHAR       ;переход, если нет символа
            CMP   AL,0          ;проверка на расширенный код
            JE    EXT_COD       ;если да, то на специальную процедуру
...                             ;иначе - в AL код ASCII
EXT_COD: INT  21H               ;получаем номер расширенного кода
...                             ;номер кода в AL
```

3.1.6. Получение строки символов

И Бейсик, и MS-DOS предоставляют возможности для выполнения процедур приема строки символов. Они автоматически повторяют процедуры ввода одного символа, описанные ранее, ожидая ввода возврата каретки, сигнализирующего об окончании строки. Конечно, должна быть отведена память, достаточная для приема всех символов строки, и должна записываться длина каждой строки для того, чтобы отделить одну строку от другой. Это делается с помощью дескрипторов строки, которые состоят из одного или более байтов, содержащих адрес и/или длину строки. В Бейсике первые два байта дескриптора строки содержат адрес строки, а сами дескрипторы хранятся в массиве отдельно от строк. Длина строки хранится в третьем байте 3-байтового дескриптора. С дру-

гой стороны, DOS хранит длину строки прямо в начале самой строки, и для программы достаточно знать положение строки в памяти.

Высокий уровень

Бейсик может принимать строку как с автоматическим эхом на экране, так и без него. Более просто делается ввод с эхом, так как он выполняется встроенной функцией ввода строки INPUT. INPUT автоматически собирает вводимые символы, выводя их на экран по мере получения. При нажатии клавиши «Enter» ввод завершается, и значение строки присваивается указанной переменной (посылаемый клавишей «Enter» код ASCII 13 не добавляется к строке). INPUT допускает возможность редактирования строки, предоставляемую DOS, поэтому опечатки могут быть исправлены перед вводом строки. INPUT принимает числа в виде строки и автоматически преобразует их в числовую форму, если для ввода будет указано имя числовой переменной. Наконец, INPUT может выдавать на экран строку, запрашивающую пользователя о требуемой информации. Такая строка может быть длиной до 254 символов. Если ее длина больше, то лишние символы игнорируются. Основная форма этого оператора - INPUT "запрос", имя_переменной. Полное описание этого оператора см. в руководстве по Бейсику.

```
110 INPUT "Enter your name: ",NAME$   'принимает имя как строку
120 INPUT "Enter your age: ",AGE%     'принимает возраст как число
```

Оператор INPUT неприменим, если в вводимой строке могут встретиться расширенные коды, такие, как коды управления курсором в полноэкранном текстовом редакторе. В этом случае требуется использовать функцию INKEY$ для приема каждого символа, затем проверять ввод на расширенные коды, выделять символы управления курсором, такие, как возврат каретки, и только после этого выводить на экран те символы, которые следует выводить. При этом управляющие символы также включаются по одному в конец строковой переменной. Текстовые файлы представляют собой набор таких строковых переменных. В параграфе [3.1.8] вы найдете процедуру ввода с клавиатуры, в которой функция INKEY$ используется указанным образом.

Средний уровень

Функция 0AH прерывания 21H позволяет вводить строку длиной до 254 символов, выдавая эхо на терминал. Эта процедура

продолжает ввод поступающих символов до тех пор, пока не нажата клавиша возврата каретки. DS:DX указывает на адрес памяти, куда должна быть помещена строка. При входе первый байт в этой позиции должен содержать число байтов, отводимых для этой строки. После того как строка введена, второй байт даст число реально введенных символов. Сама строка начинается с третьего байта.

Надо отвести достаточно памяти для строки нужной длины плюс два байта для дескриптора строки и один добавочный байт для возврата каретки. Когда вы устанавливаете максимальную длину строки в первом байте, не забудьте добавить 1 для возврата каретки. Код возврата каретки - ASCII 13 - вводится как последний символ строки, но он не учитывается в результате, который функция помещает во второй байт дескриптора строки. Таким образом, для получения 50-символьной строки надо отвести 53 байта памяти и поместить в первый байт ASCII 51. После ввода 50 символов второй байт будет содержать ASCII 50, а 53-й байт отведенной памяти - ASCII 13.

```
;---в сегменте данных
STRING  DB   53 DUP(?)    ;область для строки 50 символов
;---получение строки с клавиатуры
        LEA  DX,STRING    ;DS:DX указывают на адрес строки
        MOV BX,DX         ;пусть BX тоже указывает на строку
        MOV AL,51         ;установка длины строки ( + 1 для CR)
        MOV [BX],AL       ;посылаем в 1-й байт дескриптора
        MOV AH,0AH        ;номер функции
        INT  21H          ;получаем строку
;---проверка длины строки
        MOV AH,[BX] + 1   ;теперь длина в AH
```

В этой процедуре можно использовать возможности редактирования строки MS-DOS. Нажатие клавиши «Backspace» или "стрелка-влево" удаляет символ с экрана без помещения его в память. Работает клавиша табуляции, расширенные коды игнорируются, пустые строки допускаются (имеется в виду возврат каретки, которому не предшествует другой символ). На терминале при достижении правого края вводимая строка переносится на следующую строку дисплея, а при достижении правого нижнего угла экран сдвигается на строку вверх. Когда вводится больше символов, чем отведено места для строки, лишние символы игнорируются и включается гудок динамика.

MS-DOS обеспечивает и другой способ получения строки, при котором эхо не выводится на терминал. Функция 3FH прерывания 21H - это функция ввода общего назначения, которая чаще всего

применяется при дисковых операциях. Она требует предопределенного дескриптора файла (file handle), который является кодовым числом, используемым операционной системой для обозначения устройства ввода/вывода. Для клавиатуры используется дескриптор 0 и он должен быть помещен в BX. Поместите в DS:DX адрес, по которому должна находиться строка, а в CX - максимальную длину строки и вызовите функцию:

```
;---чтение строки без эха
MOV  AH,3FH          ;номер функции
MOV  BX,0            ;номер дескриптора файла
LEA  DX,STR_BUFER    ;указатель на буфер ввода строки
MOV  CX,100          ;максимальная длина строки
INT  21H             ;ждем ввода
```

Ввод строки завершается нажатием клавиши возврата каретки и DOS добавляет в конец строки два символа: возврат каретки и перевод строки (ASCII 13 и ASCII 10). Из-за этих добавочных символов при указании длины строки в 100 символов она может занимать до 102 байт памяти. Длина введенной строки возвращается в AX, и это значение включает два символа-ограничителя.

3.1.7. Проверка/установка статуса клавиш-переключателей

Два байта, расположенные в ячейках памяти 0040:0017 и 0040:0018, содержат биты, отражающие статус клавиши сдвига и других клавиш-переключателей следующим образом:

	Бит	Клавиша	Значение, когда бит = 1
0040:0017	7	Insert	режим вставки включен
	6	CapsLock	режим CapsLock включен
	5	NumLock	режим NumLock включен
	4	ScrollLock	режим ScrollLock включен
	3	Alt	клавиша нажата
	2	Ctrl	клавиша нажата
	1	левый Shift	клавиша нажата
	0	правый Shift	клавиша нажата
0040:0018	7	Insert	клавиша нажата
	6	CapsLock	клавиша нажата
	5	NumLock	клавиша нажата
	4	ScrollLock	клавиша нажата
	3	Ctrl-NumLock	режим Ctrl-NumLock включен

(остальные биты не используются)

Прерывание клавиатуры немедленно обновляет эти биты статуса, как только будет нажата одна из клавиш-переключателей, даже если не было считано ни одного символа из буфера клавиатуры. Это верно и для клавиши «Ins», которая единственная из этих 8 клавиш помещает код в буфер (установка статуса «Ins» меняется, даже если в буфере нет места для символа). Отметим, что бит 3 по адресу 0040:0018 устанавливается в 1, когда действует режим задержки «Ctrl-NumLock»; поскольку в этом состоянии программа приостановлена, этот бит несуществен.

Прерывание клавиатуры проверяет состояние статусных битов перед тем, как интерпретировать нажатые клавиши, поэтому когда программа меняет один из этих битов, эффект тот же, что и при физическом нажатии соответствующей клавиши. Вы можете захотеть установить состояние клавиш «NumLock» и «CapsLock», чтобы быть уверенным, что ввод будет требуемого вида. Наоборот, ваша программа может нуждаться в чтении статуса этих клавиш, например, для того, чтобы вывести текущий статус на экран. Отметим, что клавиатура AT правильно устанавливает световые индикаторы состояния клавиш, даже если их статусы переключены программно.

Высокий уровень

В данном примере клавиша «NumLock» переводится в режим, когда клавиши дополнительной клавиатуры используются для перемещения курсора в результате сбрасывания бита 5 по адресу 0040:0017 в 0. Это достигается за счет операции логического "И" значения, расположенного по этому адресу, с числом 223 (цепочка битов 11011111B, описание логики битовых операций см. в приложении Б). Результат помещается в байт статуса. В примере затем восстанавливается значение этого бита в 1 посредством логического "ИЛИ" с 32 (00100000B).

```
100 DEF SEG = &H40                    'устанавливаем сегмент на область
110 STATUSBYTE = PEEK(&H17)           'BIOS и берем байт статуса
120 NEWBYTE = STATUSBYTE AND 223      'обнуляем бит 5
130 POKE(&H17,NEWBYTE)                'посылаем новое значение статуса
```

Включение этого бита:

```
120 NEWBYTE = STATUSBYTE OR 32        'устанавливаем бит 5
130 POKE(&H17,NEWBYTE)                'посылаем новое значение статуса
```

Строки 110-130 могут быть уплотнены:

```
110 POKE(&H417,PEEK(&H417)AND 223)
```

или

```
110 POKE(&H417,PEEK(&H417)OR 223)
```

Средний уровень

Функция 2 прерывания 16H предоставляет доступ к одному (только к одному) из байтов статуса. Это байт по адресу 0040:0017, который содержит больше полезной информации. Байт возвращается в AL.

```
;---проверка статуса клавиши вставки
MOV  AH,2            ;номер функции
INT  16H             ;получаем байт статуса
TEST AL,10000000B    ;проверяем бит 7
JZ   INSERT_OFF      ;если 0, то INSERT выключен
```

Низкий уровень

В данном примере устанавливается режим вставки за счет установки бита 7 байта статуса по адресу 0040:0017 (который адресуется как 0000:0417).

```
SUB  AX,AX           ;устанавливаем добавочный сегмент на
MOV  ES,AX           ;начало памяти
MOV  AL,10000000B    ;готовим бит 7 к установке
OR   ES:[417H],AL    ;меняем байт статуса
```

3.1.8. Написание процедуры ввода с клавиатуры общего назначения

Система кодов, используемых клавиатурой, не поддается простой интерпретации. Коды могут иметь длину 1 или 2 байта, и нет простого соответствия между длиной кода и тем, служит ли он для обозначения символа или для управления оборудованием. Не все комбинации клавиш выдают уникальный код, поэтому необходимы добавочные усилия, чтобы различить их. Ни коды

ASCII, ни расширенные коды не упорядочены так, чтобы была возможна их простая группировка и проверка ошибок. Другими словами, процедура ввода с клавиатуры общего назначения требует хлопотливого программирования.

Здесь приведены примеры на Бейсике и показано использование прерывания 16H, а также как свести вместе большую часть информации, приведенной в данной главе. Общий алгоритм представлен на рис. 3.3.

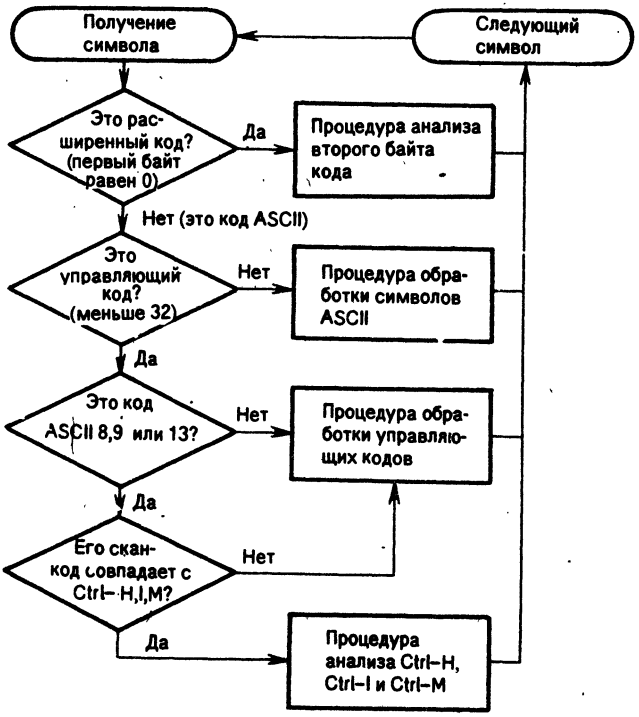


Рис. 3.3. Блок-схема процедуры ввода общего назначения

Высокий уровень

Процедура обработки ввода с клавиатуры, написанная на Бейсике, обеспечивает то же, что и ассемблерная процедура, за одним

исключением. Функция INKEY$ не предоставляет доступа к сканкодам. Это означает, что вы не можете сказать, получены ли коды ASCII 8, 9, 13 и 27 от нажатия клавиш «BackSpace», «Tab», «Enter» и «Escape» или через Ctrl + H, + I, + M и + [. Различие может быть установлено проверкой бита статуса клавиши «Ctrl» по адресу 0040:0017 в момент нажатия клавиши. Но этот метод не будет работать, если введенный символ был запасен в буфере клавиатуры в течение некоторого времени.

```
100 C$ = INKEY$:IF C$ = "" THEN 100  'получение символа
110 IF LEN(C$) = 2 THEN 700          'если расширенный, то на 700
120 C = ASC(C$)                      'иначе берем номер кода ASCII
130 IF C<32 THEN 300                 'если управляющий, то на 300
140 IF C<65 OR C>123 THEN 100        'принимаем только символы
150 '''пишущей машинки и делаем с ними, что хотим,
```

например:

```
160 S$ = S$ + C$                     'добавляем символ к строке
170 PRINT C$;                        'выводим его на экран
180 '''... и т.д.
190 GOTO 100                         'на ввод следующего символа
.
.
300 '''процедура обработки управляющих кодов ASCII
310 DEF SEG = 0                      'указываем на начало памяти
320 REGISTER = PEEK(&H417)           'берем регистр статуса
330 X = REGISTER AND 4               'X = 4, когда нажата Ctrl
340 IF X = 0 THEN 500                'если не нажата, то на 500
350 '''если это комбинация Ctrl-буква, то делаем, что хотим,
360 IF C = 8 THEN GOSUB 12000        'например, переходим на
370'''процедуру вывода экрана помощи и т.д.
380 GOTO 100                         'на ввод следующего символа
.
.
500 '''процедура обработки 4 клавиш: декодирует коды ASCII 8,
510 '''9, 13 и 27, когда клавиша «Ctrl» не нажата
520 IF C = 8 THEN GOSUB 5000         'обработка «BackSpace»
530 IF C = 9 THEN GOSUB 6000         'обработка «Tab»
540 IF C = 13 THEN GOSUB 7000        'обработка «CR»
```

```
550 IF C = 27 THEN GOSUB 8000        'обработка «Esc»
560 GOTO 100                         'на ввод следующего символа

.
.

700 '''процедура обработки расширенных кодов
710 C$ = RIGHT$(C$,1)                'берем только 2-й байт C$
720 C = ASC(C$)                      'переводим в числовую форму
730 '''в C - расширенный код - делаем с ним, что хотим,
```

например,

```
740 IF C<71 OR C>81 THEN 100         'берем только управление курсором
750 IF C = 72 THEN GOSUB 3500        'обработка "курсор-вверх"
760 '''... и т.д. 770 GOTO 100       'на ввод следующего символа
```

Средний уровень

Этот пример отличается от предыдущего методом распознавания четырех частных случаев Ctrl + H, + I, + M и + [. Здесь, когда встает вопрос о том, возник ли указанный код при нажатии одной клавиши или в комбинации с клавишей «Ctrl», проверяется скан-код. Этот метод более правилен, чем проверка бита статуса, так как скан-код запоминается в буфере клавиатуры, а установка бита статуса может быть изменена.

```
;---получение кода нажатой клавиши и определение его типа
NEXT:   MOV  AH,0          ;функция ввода с клавиатуры BIOS
        INT  16H           ;получаем введенный код
        CMP  AL,0          ;проверка на расширенный код
        JE   EXT_COD       ;если да, то на специальную процедуру
        CMP  AL,32         ;проверка на управляющий символ
        JL   C_COD         ;если да, то на специальную процедуру
        CMP  AL,65         ;если символ не входит в набор пишу-
        JL   NEXT          ;щей машинки, то берем следующий
        CMP  AL,123
        JL   NEXT
;---теперь обрабатываем символ в AL
        STOSB              ;запоминаем символ по адресу ES:DI
        MOV  AH,2          ;функция вывода символа на экран
        MOV  DL,AL         ;помещаем символ в DL перед выводом
```

```
            INT   21H          ;выводим его на экран
.
            JMP   NEXT         ;переходим к следующему символу
;---анализируем управляющие коды
C_COD:      CMP   AL,13        ;код ASCII 13?
            JNE   TAB          ;если нет, то следующая проверка
            CMP   AH,28        ;иначе проверяем скан-код «CR»
            JNE   C_M          ;если нет, то было Ctrl + M
            CALL  CARRIAGE_RET     ;обработка возврата каретки
            JMP   NEXT         ;переход к следующему символу
C_M:        CALL  CTRL_M       ;обработка Ctrl + M
            JMP   NEXT         ;переход к следующему символу
TAB:        CMP   AL,9         ;проверка на табуляцию...
.
            CMP   AL,10        ;затем проверка других
.
REJECT:     JMP   NEXT         ;переход к следующему символу
;---анализ расширенных кодов (2-й байт кода в AH):
EXT_COD:    CMP   AH,71        ;проверка нижней границы
            JL    REJECT       ;если меньше, то следующий символ
            CMP   AH,81        ;проверка верхней границы
            JL    REJECT       ;если больше, то следующий символ
;---AH содержит символ управления курсором, анализируем его
            CMP   AH,72        ;"курсор-вверх"?
            JE    C_U          ;если да, то на процедуру
            CMP   AH,80        ;"курсор-вниз"?
            JE    C_D          ;если да, то на процедуру
.
.
C_U:        CALL  CURS_UP      ;вызов соответствующей процедуры
            JMP   NEXT         ;переход к следующему символу
C_D:        CALL  CURS_DOWN    ;вызов соответствующей процедуры
            JMP   NEXT         ;переход к следующему символу
```

3.1.9. Перепрограммирование прерывания клавиатуры

Когда микропроцессор клавиатуры помещает скан-код в порт A микросхемы 8255 (адрес порта 60H, см. [1.1.1]), вызывается прерывание 9. Задача этого прерывания - преобразовать скан-код символа, основываясь на состоянии клавиш-переключателей, и помес-

тить его в буфер клавиатуры. (Если скан-код соответствует клавише-переключателю, то в буфер клавиатуры не пишется ничего, исключение составляет клавиша «Ins», вместо этого прерывание изменяет байты статуса, расположенные в области данных BIOS [3.1.7].) Прерывания "ввода с клавиатуры" DOS и BIOS на самом деле - всего лишь прерывания "ввода из буфера клавиатуры". Они не распознают нажатия клавиш. Точнее, они читают интерпретацию введенных клавиш, которую обеспечило прерывание 9. Заметим, что PCjr использует специальную процедуру (INT 48H) для преобразования ввода от его 62 клавиш к 83-клавишному протоколу, используемому другими IBM PC. Результат этой процедуры передается прерыванию 9, которое выполняет свою работу как обычно. Прерыванием 49H PCjr обеспечивает специальные неклавишные скан-коды, которые потенциально могут устанавливаться периферийными устройствами, использующими инфракрасную (беспроволочную) связь с клавиатурой.

Только при весьма необычном применении имеет смысл перепрограммирование этого прерывания, особенно учитывая, что MS-DOS позволяет перепрограммировать любую клавишу клавиатуры [3.2.6]. Если вам все же придется перепрограммировать прерывание 9, эта глава даст вам основы для старта. Сначала надо прочитать [1.2.3], чтобы понять, как программируются прерывания. В прерывании клавиатуры можно выделить три основных шага:

1. Прочитать скан-код и послать клавиатуре подтверждающий сигнал.

2. Преобразовать скан-код в номер кода или в установку регистра статуса клавиш-переключателей.

3. Поместить код клавиши в буфер клавиатуры.

В момент вызова прерывания скан-код будет находиться в порте A. Поэтому сначала надо этот код прочитать и сохранить на стеке. Затем используется порт B (адрес 61H), чтобы быстро послать сигнал подтверждения микропроцессору клавиатуры. Надо просто установить бит 7 в 1, а затем сразу вернуть его в 0. Заметим, что бит 6 порта B управляет сигналом часов клавиатуры. Он всегда должен быть установлен в 1, иначе клавиатура будет выключена. Эти адреса портов применимы и к AT, хотя он и не имеет микросхемы интерфейса с периферией 8255.

Сначала скан-код анализируется, была ли клавиша нажата (код нажатия) или отпущена (код освобождения). На всех машинах, кроме AT, код освобождения индицируется установкой бита 7 скан-кода в 1. Для AT, у которого бит 7 всегда равен 0, код освобождения состоит из двух байтов: сначала 0F0H, а затем скан-код. Все коды освобождения отбрасываются, за исключением клавиш-переключателей, для которых делаются соответствующие изменения в байтах их статуса. С другой стороны, все коды нажатия

обрабатываются. При этом снова могут изменяться байты статуса клавиш-переключателей. В случае же символьных кодов надо проверять байты статуса, чтобы определить, например, что скан-код 30 соответствует нижнему или верхнему регистру буквы A.

После того как введенный символ идентифицирован, процедура ввода с клавиатуры должна найти соответствующий ему код ASCII или расширенный код. Приведенный пример слишком короток, чтобы рассмотреть все ситуации. В общем случае скан-коды сопоставляются с элементами таблицы данных, которая анализируется инструкцией XLAT. XLAT принимает в AL число от 0 до 255, а возвращает в AL 1-байтовое значение из 256-байтовой таблицы, на которую указывает DS:BX. Таблица может находиться в сегменте данных. Если в AL находился скан-код 30, то туда будет помещен из таблицы байт номер 30 (31 байт, так как отсчет начинается с нуля). Этот байт в таблице должен быть установлен равным 97, что дает код ASCII для "a". Конечно, для получения заглавной A нужна другая таблица, к которой обращение будет происходить, если статус сдвига установлен. Или заглавные буквы могут храниться в другой части той же таблицы, но в этом случае к скан-коду надо будет добавлять смещение, определяемое статусом клавиш-переключателей.

Наконец, номера кодов должны быть помещены в буфер клавиатуры. Процедура должна сначала проверить, имеется ли в буфере место для следующего символа. В [3.1.1] показано, что этот буфер устроен как циклическая очередь. Ячейка памяти 0040:001A содержит указатель на голову буфера, а 0040:001C - указатель на его хвост. Эти словные указатели дают смещение в области данных BIOS (которая начинается в сегменте 40H) и находятся в диапазоне от 30 до 60. Новые символы вставляются в ячейки буфера с более старшими адресами, а когда достигнута верхняя граница, следующий символ переносится в нижний конец буфера. Когда буфер полон, указатель на его хвост на 2 меньше указателя на голову, за исключением ситуации, когда указатель на голову равен 30 (начало области буфера), а в этом случае буфер полон, когда указатель на его хвост равен 60.

Для вставки символа в буфер надо поместить его в позицию, на которую указывает хвост буфера и затем увеличить указатель хвоста на 2; если указатель хвоста был равен 60, то надо изменить его значение на 30. Вот и все. Схема прерывания клавиатуры показана на рис. 3.4.

Низкий уровень

Эффективная процедура требует глубокого продумывания. В этом примере представлены элементарные основы: принимаются

только буквы нижнего и верхнего регистров, причем все они загружены в одну таблицу, где буквы верхнего регистра находятся на 100 байт выше, чем их младшие братья. Анализируется только левая клавиша сдвига, и текущее состояние клавиши «CapsLock» игнорируется.

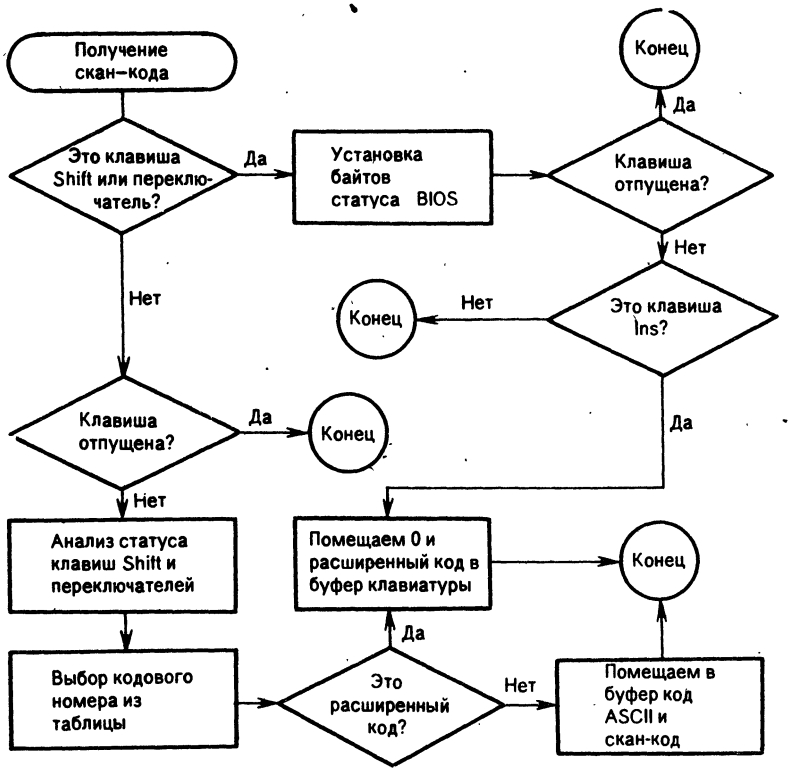


Рис. 3.4. Прерывание клавиатуры

```
;---в сегменте данных
TABLEDB   16 DUP(0)              ;пропускаем первые 16 байт
      DB  'qwertyuiop',0,0,0,0   ;верхний ряд клавиатуры
      DB  'asdfghjkl',0,0,0,0,0  ;средний ряд клавиатуры
      DB  'zxcvbnm'              ;нижний ряд клавиатуры
      DB  16 DUP(0)              ;пропуск до верхнего регистра
```

```
        DB    'QWERTYUIOP',0,0,0,0 ;те же символы на верхнем
        DB    'ASDFGHJKL',0,0,0,0,0 ;регистре
        DB    'ZXCVBNM'
;---в начале программы устанавливаем прерывание
        CLI                        ;запрет прерываний
        PUSH DS                    ;сохраняем регистр
        MOV  AX,SEG N_KEY          ;DS:DX должны указывать на
        MOV  DS,AX                 ;процедуру обработки
        MOV  DX,OFFSET N_KEY       ;прерывания
        MOV  AL,9                  ;номер вектора прерывания
        MOV  AH,25H                ;номер функции DOS
        INT  21H                   ;меняем вектор прерывания
        POP  DS                    ;восстанавливаем регистр
        STI                        ;разрешаем прерывания
```

Программа продолжается, оставаясь резидентной [1.3.4].

```
;---это само прерывание клавиатуры
N_KEY   PROC FAR                   ;сохраняем все изменяемые
        PUSH AX                    ;регистры
        PUSH BX
        PUSH CX
        PUSH DI
        PUSH ES
;---получаем скан-код и посылаем сигнал подтверждения
        IN   AL,60H                ;получаем скан-код из порта A
        MOV  AH,AL                 ;помещаем копию в AH
        PUSH AX                    ;сохраняем скан-код
        IN   AL,61H                ;читаем состояние порта B
        OR   AL,10000000B          ;устанавливаем бит 7
        OUT  61H,AL                ;посылаем измененный байт в порт
        AND  AL,01111111B          ;сбрасываем бит 7
        OUT  61H,AL                ;возвращаем состояние порта B
;---ES должен указывать на область данных BIOS
        MOV  AX,40H                ;устанавливаем сегмент
        MOV  ES,AX
        POP  AX                    ;возвращаем скан-код из стека
;---проверка клавиши сдвига
        CMP  AL,42                 ;нажат левый сдвиг?
        JNE  KEY_UP                ;нет - смотрим следующее
        MOV  BL,1                  ;да - изменяем бит статуса
```

```
        OR    ES:[17H],BL       ;меняем прямо регистр статуса
        JMP   QUIT              ;выход из процедуры
KEY_UP: CMP   AL,170            ;левый сдвиг отпущен?
        JNE   NEXTKEY           ;нет - смотрим следующее
        MOV   BL,11111110B      ;да - меняем бит статуса
        AND   ES:[17H],BL       ;меняем прямо регистр статуса
        JMP   QUIT              ;выход из процедуры
NEXTKEY:                        ;просмотр других переключателей
;---это символьная клавиша - интерпретируем скан-код
        TEST  AL,10000000B      ;код освобождения клавиши?
        JNZ   QUIT              ;да - выходим из процедуры
        MOV   BL,ES:[17H]       ;иначе,берем байт статуса
        TEST  BL,00000011B      ;клавиша сдвига нажата?
        JZ    CONV              ;нет - уходим дальше
        ADD   AL,100            ;да - значит, заглавная буква
CONV:   MOV   BX,OFFSET TABLE   ;готовим таблицу
        XLAT  TABLE             ;преобразуем скан-код в ASCII
        CMP   AL,0              ;возвращен 0?
        JE    QUIT              ;если да, то на выход
;---код клавиши готов, проверяем, не полон ли буфер клавиатуры
        MOV   BX,1AH            ;смещение указателя на голову
        MOV   CX,ES:[BX]        ;получаем его значение
        MOV   DI,ES:[BX] + 2    ;получаем указатель на хвост
        CMP   CX,60             ;голова на вершине буфера?
        JE    H_END             ;да - переходим к специальному случаю
        INC   CX                ;увеличиваем указатель на голову
        INC   CX                ;на 2
        CMP   CX,DI             ;сравниваем с указателем на хвост
        JE    QUIT              ;если равны, то буфер полон
        JMP   GO_AH             ;иначе,вставляем символ
H_END:  CMP   DI,30             ;проверка специального случая
        JE    QUIT              ;если буфер полон, то выход
;---буфер не полон - вставляем в него символ
GO_AH:  MOV   ES:[DI],AL        ;помещаем символ в позицию хвоста
        CMP   DI,60             ;хвост в конце буфера?
        JNE   NO_W              ;если нет, то добавляем 2
        MOV   DI,28             ;иначе,указатель хвоста = 28 + 2
NO_W:   ADD   DI,2              ;получаем новое значение хвоста
        MOV   ES:[BX] + 2,DI    ;посылаем его в область данных
```

```
;---завершение прерывания
QUIT:   POP  ES              ;восстанавливаем изменяемые
        POP  DI              ;регистры
        POP  CX
        POP  BX
        POP  AX
        MOV  AL,20H          ;выдаем сигнал об окончании
        OUT  20H,AL          ;аппаратного прерывания
        IRET ;возврат из прерывания
N_KEY   ENDP
```

3.2. Доступ к отдельным клавишам

Процедура обработки нажатия клавиши должна проверять массу различных типов клавиш и условий, поскольку как одно-, так и двухбайтовые коды могут появляться в комбинации с клавишами-переключателями. Не все клавиши логически сгруппированы по типу кода, который им соответствует. Например, клавиша «Backspace» генерирует однобайтовый код ASCII, а клавиша «Delete» - двухбайтовый расширенный код. Клавиша «Ctrl» генерирует однобайтовый код в сочетании с алфавитными клавишами и двухбайтовый код в остальных случаях. Эти нерегулярности возникают из-за ограниченности набора ASCII: прерывание клавиатуры следует соглашениям ASCII, когда это возможно, но когда невозможно - выдает свои (расширенные) коды.

В данном разделе перечислены различные группы клавиш, даны их коды и указаны встречающиеся аномалии. В большинстве случаев эта информация доступна в менее удобном виде из таблиц кодов ASCII и расширенных кодов, приведенных в разделе 3 этой главы. Здесь обсуждаются также специальные свойства, приписываемые отдельным клавишам Бейсиком, а также специальная обработка для интерпретации отдельных клавиш (таких, как «Backspace»), применяемая в прерываниях DOS.

3.2.1. Использование клавиш «Backspace», «Enter», «Escape» и «Tab»

Клавиши «Backspace», «Enter», «Escape» и «Tab» - единственные четыре несимвольные клавиши, генерирующие однобайтовые

коды ASCII. Эти коды содержатся в наборе управляющих кодов [7.1.9], которые занимают первые 32 кода в наборе ASCII. Эти четыре кода могут быть получены также комбинацией буквенных клавиш с клавишей «Ctrl»:

ASCII 8	Backspace	Ctrl + H
ASCII 9	Tab	Ctrl + I
ASCII 13	Enter	Ctrl + M
ASCII 27	Escape	Ctrl + [

В [3.2.2] показано, как различать нажатие одной клавиши и комбинацию с клавишей «Ctrl». Отметим, что обратная табуляция, производимая при комбинации клавиш «Shift» + «Tab», выдает расширенный код 0;15.

Некоторые из прерываний обработки ввода с клавиатуры автоматически интерпретируют эти четыре специальных кода. В Бейсике функция INPUT реагирует на «Backspace», «Tab» и «Enter». Функция INKEY$ не интерпретирует ни один из управляющих кодов, поскольку у нее нет автоматического эха на экран. Всю работу должна выполнять ваша программа. Напомним, что для управления движением курсора Бейсик предоставляет функцию TAB. Прерывания BIOS и DOS, которые выдают эхо на терминал, интерпретируют также клавиши «Backspace» и «Tab». После того как эти коды интерпретируются соответствующим образом, коды ASCII все равно появляются в AL, где они могут быть включены в строку символов или игнорированы в зависимости от того, что требуется.

3.2.2. Использование клавиш-переключателей «Shift», «Ctrl» и «Alt»

Три типа клавиш-переключателей заставляют другие клавиши клавиатуры генерировать различные коды. Как правило, такие комбинации генерируют расширенные коды. Но в двух случаях они дают коды ASCII: (1) когда используется клавиша «Shift» с клавишами алфавитно-цифровых символов и (2) нажатие клавиш в комбинации от Ctrl-A до Ctrl-Z дает ASCII коды от 1 до 26. Все остальные комбинации позволяют получить расширенные коды, перечисленные в [3.3.5]. PCjr имеет несколько исключений, которые обсуждаются ниже.

Недопустимые комбинации клавиш не производят кода вообще. За исключением специальных комбинаций с «Ctrl-Alt», одновременное нажатие нескольких переключателей приводит к тому, что

только один из них становится эффективным, причем приоритет у «Alt», затем «Ctrl» и затем «Shift». В [3.1.7] показано, как проверить, нажата ли в данный момент клавиша-переключатель, а в [3.2.3] - как использовать клавишу «ScrollLock» в качестве переключателя с любой другой клавишей клавиатуры. Другие комбинации с клавишами-переключателями можно сделать допустимыми, только полностью переписав прерывание клавиатуры, которое заменило бы прерывание BIOS [3.1.9].

Существует проблема, связанная с некоторыми комбинациями с клавишей «Ctrl», такими, как Ctrl + H, I, M и [, поскольку они генерируют коды ASCII, идентичные тем, которые генерируют клавиши «Backspace», «Tab», «Enter» и «Escape». В [3.1.8] показано, как программа на ассемблере может, проверив скан-коды, определить, была ли нажата управляющая клавиша или комбинация алфавитно-цифровой клавиши с «Ctrl» (скан-код находится в AH, когда мы получаем код нажатой клавиши через прерывание 16H). К сожалению, программы на Бейсике лишены такой возможности. В подобном случае программа может попытаться различить эти две возможности, анализируя состояние регистра статуса. Если бит 2 байта статуса по адресу 0040:0017 установлен, то клавиша «Ctrl» нажата. Этот метод работает только в тот момент, когда происходит нажатие клавиши, но не тогда, когда вы берете символ из буфера клавиатуры через некоторое время.

Клавиатура PCjr имеет только 63 клавиши по сравнению с 83 для IBM PC или XT и 84 для AT. Некоторые комбинации клавиш-переключателей служат для имитации недостающих клавиш (комбинации с использованием функциональных клавиш приведены в [3.2.5]):

Комбинация клавиш PCjr	Эквиваленты в PC/XT/AT
Alt + Fn + 0-9	0-9 (скан-коды дополнительной цифровой клавиатуры)
Alt + /	\
Alt + `	`
Alt + [	|
Alt +]	~
Alt + .	* (скан-код, как от клавиши PrtSc)
Shift + Del	. (скан-код, как от дополнительной клавиатуры)

Клавиатура PCjr допускает также следующие уникальные комбинации с участием клавиш-переключателей:

Fn + Shift + Esc	переключает цифровые клавиши в функциональные
Ctrl + Alt + CapsLock	переключает звуковое подтверждение нажатия клавиши
Ctrl + Alt + Ins	запускает диагностику
Ctrl + Alt + CursorLeft	сдвигает экран влево
Ctrl + Alt + CursorRight	сдвигает экран вправо

3.2.3. Использование клавиш-переключателей: «NumLock», «CapsLock», «Ins» и «ScrollLock»

За исключением клавиши «Ins», все остальные клавиши-переключатели не производят кода, который помещался бы в буфер клавиатуры. Вместо этого они изменяют состояние двух байтов статуса, которые расположены в области данных BIOS по адресам 0040:0017 и 0040:0018. Прерывание клавиатуры проверяет установку этих байтов перед тем, как присвоить код введенному символу. Ваши программы имеют доступ к регистрам статуса и могут изменить установку любой из клавиш-переключателей, как объяснено в [3.1.7].

Другие биты регистра статуса показывают, нажата ли данная клавиша-переключатель в текущий момент. Это свойство позволяет программе использовать клавиши-переключатели в качестве клавиш сдвига. Возможны потенциальные применения этого, пока не создано новых кодов клавиш. Например, «ScrollLock» может быть использована для того, чтобы добавить добавочный набор комбинаций сдвиг + функциональная клавиатура. Программа, которая будет получать код обычной функциональной клавиши, должна проверять, нажата ли клавиша «ScrollLock», и соответственно интерпретировать нажатие клавиши. Отметим, что любая из клавиш «Shift» обращает текущую установку клавиши «NumLock».

Клавиша «Ins» помещает в буфер клавиатуры код 0;82, который ваша программа может прочитать в любой момент. Однако установка для «Ins» в байтах регистра статуса меняется немедленно. Даже если в буфере нет места для кода «Ins», в регистре статуса при нажатии клавиши вносятся изменения. Как «Ins», так и «ScrollLock» не влияют на другие клавиши клавиатуры (в отличие от «NumLock» и «CapsLock»). Вы можете приписать им любую роль. Техническое руководство IBM утверждает, что клавиша «ScrollLock» должна использоваться для переключения между состояниями, когда нажатие клавиши перемещения курсора приводит к сдвижке экрана, а не к передвижению курсора.

Конечно, вы можете создать все нужные для вашей программы клавиши-переключатели, просто назначив клавиши для этой цели. Хотя у вас нет готовых регистров статуса, но вы можете создать переменную, значение которой -1 соответствует включенному состоянию вашего переключателя, а значение 0 - выключенному. Например, используем клавишу F10 для включения и выключения переменной Clock:

```
100 '''переключение статуса переменной
110 CLOCK = -1                    'начинаем с включенным состоянием
120 IF X« = 100 THEN NOT CLOCK    'переключаем переменную
```

3.2.4. Использование цифровой дополнительной клавиатуры и клавиш перемещения курсора

Для IBM PC и XT дополнительная цифровая клавиатура включает цифровые клавиши, клавиши «Ins» и «Del», а также клавиши «+» и «-». На AT добавляется клавиша «Sys Rec», в то время как PCjr имеет только 4 клавиши перемещения курсора (остальные могут быть эмулированы специальными комбинациями с клавишами «Shift» и «Fn», описанными в [3.2.2] и [3.2.5]). Клавиша «NumLock» переключает дополнительную клавиатуру между цифрами и клавишами управления курсором. Клавиши «Ins» и «Del» работают, только если режим «NumLock» включен, т.е. дополнительная клавиатура выдает цифры. Клавиши «+» и «-» выдают одни и те же коды независимо от установки режима «NumLock».

Цифровые клавиши дополнительной клавиатуры выдают в точности те же однобайтовые коды, что и цифровые клавиши верхнего ряда основной клавиатуры, т.е. коды ASCII от 48 до 57 для цифр от 0 до 9. Это верно и для клавиш «+» и «-». Программисты на ассемблере могут определить, какая из двух клавиш нажата по скан-коду клавиши, который находится в AH при возврате как из прерывания 16H, так и из процедур ввода одной клавиши прерывания 21H. Отметим, что любая из клавиш «Shift» переводит клавиши дополнительной клавиатуры в режим, противоположный тому, который установлен клавишей «NumLock». Установка клавиши «CapsLock» не имеет значения. Клавиша "5" в центре активна только как цифровая клавиша и в режиме перемещения курсора вообще не выдает кода.

Кроме четырех общепринятых стрелок, клавиши управления курсором включают также «Home», «End», «PgUp» и «PgDn», которые часто используются для перемещения курсора сразу на целую строку или страницу. Все они генерируют двухбайтовые расширенные коды. Эти клавиши не обеспечивают прямого конт-

роля над курсором. Они просто выдают коды, как и все другие клавиши, и задачей программиста становится преобразование этих кодов в перемещения курсора на экране.

Допустимы некоторые комбинации клавиш дополнительной клавиатуры с клавишей «Ctrl». «NumLock» должна соответствовать режиму управления курсором, чтобы эти комбинации работали. В [3.1.7] показано, как ваша программа может автоматически устанавливать режим NumLock. Вот перечень кодов клавиш дополнительной клавиатуры:

Коды ASCII:	
43	+
45	-
46	.
48-57	0-9
Расширенные коды:	
72,75,77,80	CursorUp,Left,Right & Down
71,73,79,81	Home,PgUp,End,PgDn
82,83	Ins,Del
115,116	Ctlr-cursor left, -cursor right
117,118,119,132	Ctlr-end, -PgDn, -Home, -PgUp

AT имеет 84-ю клавишу, «Sys Req», которая уникальна по своей функции. Клавиша предназначена для многопользовательских систем как способ входа в главное меню системы. Когда клавиша нажимается, в AX появляется код 8500H и выполняется прерывание 15H. При отпускании клавиши в AX появляется код 8501H и опять же выполняется прерывание 15H. BIOS AT не обрабатывает функции 84H и 85H прерывания 15H, а просто делает возврат. Но можно программно заменить вектор прерывания для 15H, чтобы он указывал на процедуру обработки клавиши «Sys Req». Такая процедура должна сначала прочитать AL, чтобы узнать, была ли клавиша нажата (AL = 0) или отпущена (AL = 1). Заметим, что прерывание 15H предоставляет ряд процедур, и некоторые из них могут потребоваться программе обработки «Sys Req». В этом случае процедура обработки «Sys Req» должна восстанавливать замененный ею вектор прерывания, и если в AH указаны функции, отличные от 84H и 85H, то надо передать управление оригинальному прерыванию 15H [1.2.4].

3.2.5. Использование функциональных клавиш

10 функциональных клавиш генерируют различные коды в сочетании с «Shift», «Ctrl» и «Alt», что обеспечивает 40 разных вариантов. Во всех случаях генерируется двухбайтовый расширенный код, в котором первый байт всегда ASCII 0, а второй байт приведен в таблице:

Коды	Клавиши
59-68	F1-F10
84-93	Shift + F1-F10
94-103	Ctrl + F1-F10
104-113	Alt + F1-F10

Слишком много комбинаций с использованием функциональных клавиш могут смущать пользователя, но если вам потребовалось еще 10 комбинаций, то можно применять сочетание «ScrollLock» + «Fn», как объяснено в [3.2.3].

Клавиатура PCjr имеет только 62 клавиши по сравнению с 83 для IBM PC и XT и 84 для AT. Некоторые комбинации с функциональными клавишами эмулируют часть недостающих клавиш согласно следующей таблице:

PCjr комбинации	Эквиваленты в PC/XT/AT
Fn + 1-0	F1-F10
Fn + B	Break
Fn + E	Ctrl + PrtSc
Fn + P	Shift + PrtSc
Fn + Q	Ctrl + NumLock
Fn + S	ScrollLock
Fn + CursorLeft	PgUp
Fn + CursorRight	PgDn
Fn + CursorUp	Home
Fn + CursorDown	End
Fn + -	(скан-код серого минуса)
Fn + =	(скан-код серого плюса)

Комбинации с клавишами-переключателями описаны в [3.2.2].

3.2.6. Перепрограммирование отдельных клавиш

Под перепрограммированием клавиши понимается способ заставить ее выдавать другой код. Но к тому времени, когда программа получает код нажатой клавиши, прерывание клавиатуры уже проинтерпретировало входящий скан-код и преобразовало его в некоторый заранее предопределенный код ASCII или расширенный код. К счастью, начиная с MS-DOS версии 2.0, система содержит средства перепрограммирования клавиш. Это средство действует, только если ввод воспринимается через функции DOS ввода с клавиатуры, функции прерывания BIOS 16H продолжают интерпретировать нажатия клавиш нормальным образом.

Перепрограммирование доступно за счет Esc-последовательностей. Короткая строка, которая начинается с символа Esc (ASCII 27), предназначается для вывода на "стандартное устройство вывода", т.е. на терминал. Но благодаря наличию кода Esc, символы даже не достигают монитора. Вместо этого такая строка заставляет MS-DOS по-другому интерпретировать клавишу, указанную в этой строке. Каждое изменение клавиши требует собственной строки, при этом один и тот же код может присваиваться какому угодно количеству клавиш.

Общий вид такой строки следующий: она начинается с кода Esc (ASCII 27), за которым идет [, затем номер кода переопределяемой клавиши, затем точка с запятой (;), затем новый номер кода, присваиваемый клавише, и, наконец, символ p. Таким образом, строка 27,'[65;97p' меняет A (ASCII 65) на a (ASCII 97). Расширенные коды записываются с указанием обоих байтов, причем за первым нулевым байтом должна стоять точка с запятой. Строка 27,'[0;68;0;83p' присваивает клавише F10 (0;68) тот же код, что и клавише Delete (0;83). Вы можете присваивать только расширенные коды, приведенные в таблице расширенных кодов [3.3.5].

Существует несколько вариантов допустимого вида строки. Во-первых, символьные клавиши могут обозначаться самим символом, заключенным в кавычки. Таким образом, строка 27,'["A";"a"p' также меняет A на a. Во-вторых, клавише может быть присвоена целая строка символов путем указания символов или их кодовых номеров в выражении. Строка 27,'["A";"A is for Apple"p' приведет к тому, что при нажатии на клавишу A в верхнем регистре будет печататься вся строчка A is for Apple. На самом деле эти Esc-последовательности - ничего более, чем строки, в которых первый код или символ указывает, какую клавишу нужно переопределить, а оставшаяся часть строки указывает, какое значение вы хотите ей придать. Помните, что номера кодов должны быть всегда разделены точкой с запятой, а символы заключены в кавычки. Коды и

символы могут быть перемешаны в любых сочетаниях. Для того чтобы такие переопределения клавиш стали возможны, необходимо, чтобы драйвер ANSI.SYS был загружен при загрузке операционной системы. В противном случае Esc-последовательности будут игнорироваться. В приложении Д показано, как это делается.

Некоторые аспекты функционирования клавиатуры программируются на PCjr и AT. Процедуры, доступные для AT, интересны в основном для системных программистов; поскольку они нужны весьма немногим и достаточно сложны, мы не будем рассматривать их здесь. При необходимости вам придется обратиться к Техническому руководству по AT. В случае PCjr прерывание BIOS 16H имеет две дополнительные функции (AH = 3 и AH = 4), первая из которых устанавливает частоту автоповтора. "Частота автоповтора" - это та частота, с которой клавиша посылает свой код, когда она постоянно держится нажатой. Вторая функция включает и выключает звуковое подтверждение нажатия клавиши. Для функции 3 надо поместить в AL 0, чтобы вернуться к частоте автоповтора, устанавливаемой по умолчанию, 1, чтобы увеличить начальную задержку перед тем, как начинается режим автоповтора, 2, чтобы уменьшить частоту автоповтора вдвое, 3, чтобы установить свойства 1 и 2 вместе, и 4, чтобы выключить автоповтор вообще. Для функции 4, поместив в AL 1, вы будете иметь звуковое подтверждение нажатия клавиши, а 0 - выключите его.

Высокий уровень

К несчастью, операторы Бейсика PRINT и WRITE не работают с Esc-последовательностями. Программы на Бейсике должны включать простые ассемблерные подпрограммы, использующие прерывания вывода MS-DOS, обсуждаемые ниже. В приложении Г показано, как включить ассемблерные процедуры в программы на Бейсике. В приведенном примере предполагается, что процедура находится в памяти, начиная с адреса 2000:0000. Операторы DATA содержат ассемблерный код. В конце строки переопределения клавиши должен быть добавлен код $.

```
100 DATA &H55, &H8B, &HEC, &H8B, &H5E, &H06, &H8B, &H57
110 DATA &H01, &HB4, &H09, &HCD, &H21, &H5D, &HCA, &H02, &H00
120 'помещаем процедуру в память по адресу 2000:0000
130 DEF SEG = &H2000                    'определяем сегмент
140 FOR N = 0 TO 16                     'процедура длиной 17 байт
150 READ Q                              'читаем байт
160 POKE N,Q                            'помещаем его в память
```

```
170 NEXT
180 ''' меняем A на a
190 Q$ = CHR$(27) + "[65;97p$"      'строка переопределения
200 ROUTINE = 0                     'указываем на строку
210 CALL ROUTINE(Q$)                'вызываем процедуру
```

Средний уровень

Используйте функцию 9 прерывания 21H для посылки строки на стандартное устройство вывода. DS:DX должны указывать на первый символ строки в памяти и строка должна завершаться символом $ (24H). Здесь F2 (0;60) переопределяется таким образом, чтобы она действовала как Del (0;83).

```
;---в сегменте данных
CH_KEY DB   27,'[0;60;0;83p$'
;---для изменения определения клавиши
        LEA  DX,CH_KEY   ;DS:DX должны указывать на строку
        MOV AH,9         ;номер функции
        INT  21H         ;переопределение клавиши
```

3.2.7. Создание макроопределений для отдельных клавиш

Макроопределение - это строка символов, которая будет выводиться при нажатии одной клавиши. Макроопределения могут быть запрограммированы в интерпретаторе Бейсика или на уровне операционной системы для уменьшения печатания. Поскольку строка может содержать управляющие коды, такие, как символ возврата каретки (ASCII 13), одно макроопределение может выполнять целый набор команд. Для ускорения разработки программ, например, можно написать макроопределение, содержащее все необходимые команды, чтобы оттранслировать и скомпоновать определенную программу.

Макроопределения, обеспечиваемые Бейсиком, работают как в бейсиковских программах, так и на командном уровне Бейсика. Например, если вы запрограммировали клавишу, чтобы при ее нажатии выводилось слово "Орангутан", то при нажатии этой клавиши функция INPUT получит всю эту строку, а цикл, включающий INKEY$, последовательно получит девять символов. С другой стороны, макроопределения, созданные на уровне операционной системы, всегда работают на командном уровне DOS, но внутри программ они будут работать, только если программа для

ввода с клавиатуры использует функции DOS. Поскольку большинство коммерческих программных продуктов используют прерывание BIOS 16H, для этих программ макроопределения не будут работать. Конечно, средства для создания макроопределений могут быть вставлены в процедуры ввода с клавиатуры. Например, чтобы позволить пользователю программы создать макроопределение для F1, запросив строку и поместив ее в MACRO1$, надо на Бейсике написать что-то вроде

```
1000 '''процедура ввода расширенного кода, в C - 2-й байт кода
1010 IF C = 59 THEN LOCATE X,Y: PRINT MACRO1$
```

Высокий уровень

Бейсик имеет встроенный механизм создания макроопределений, но он позволяет программировать только 10 функциональных клавиш, а строки должны быть не длиннее 15 символов. Бейсик рассматривает функциональные клавиши как программируемые клавиши. Оператор KEY присваивает макроопределение данной клавише. Строка KEY 5,"END" приводит к тому, что функциональная клавиша #5 будет посылать слово END в текущую позицию курсора на экране.

Символы, составляющие строку, могут вводиться как строки символов, как коды ASCII (с использованием CHR$) или как комбинация того и другого. Команды KEY 5,"A" и KEY 5,CHR$(65) эквивалентны. Для того чтобы строка сразу исполнялась, надо добавить в конце строки символ возврата каретки (ASCII 13). Команда FILES, выводящая каталог диска, исполняется после того, как вы присвоите это значение F1 командой KEY1, "FILES" + CHR$(13).

Бейсик присваивает десяти функциональным клавишам распространенные операторы Бейсика. Вы можете отменить макроопределение для данной клавиши, присвоив ей пустую строку, например, команда KEY 1,"" приведет к тому, что при нажатии F1 ничего вводиться не будет. Первые шесть символов каждой строки автоматически выводятся в нижней части экрана интерпретатором Бейсика. Вы можете управлять наличием этого вывода, используя команды KEY ON и KEY OFF. Для того чтобы вывести на экран полные определения клавиш, введите команду KEY LIST. Вот несколько примеров:

```
KEY 1,"ERASE"                    ; теперь F1 выводит "ERASE"
KEY 10,"LIST" + CHR$(13)         ; теперь F10 выдает листинг
KEY 7,""                         ; теперь F7 ничего не выдает
```

```
KEY OFF                    ; подавляет вывод внизу экрана
KEY ON                     ; включает вывод внизу экрана
KEY LIST                   ; выдает список значений 10 клавиш
```

Для создания макроопределений других клавиш в Бейсике, вы должны использовать средства MS-DOS, описанные в [3.2.6].

Средний уровень

В MS-DOS макроопределения создаются с помощью метода перепрограммирования клавиш, описанного в [3.2.6]. Единственное отличие в том, что клавише сопоставляется целая строка символов. Строка может быть введена в виде символов, заключенных в кавычки, а также в виде кодов или комбинации того и другого. Вот несколько примеров:

```
27,"[A";"SET"p"            ;присваивает SET заглавной A
27,"[ASET"p"               ;эквивалентно предыдущему
27,"[27;"dir";13p"         ;присваивает dir«CR» клавише Esc
27,"[0;59;"copy *.* b:";13p"  ;присваивает F1 команду
27,"[0;68;0;72;0;72;0;72p"    ;заставляет F10 сдвинуть курсор на
                           ;три строки вверх
```

3.2.8. Создание процедуры обработки «Ctrl-Break»

Когда вводится комбинация «Ctrl-Break», прерывание клавиатуры устанавливает флаг, указывающий, что должна быть выполнена процедура обработки «Ctrl-Break». Управление передается этой процедуре только в тот момент, когда программа обращается к функции DOS, способной распознавать этот флаг. Обычно только стандартные функции ввода/вывода MS-DOS могут распознавать этот флаг (функции от 1 до 0C прерывания 21H, за исключением функций 6 и 7). Но поместив строку BREAK = ON либо в файл AUTOEXEC.BAT, либо в CONFIG.SYS, используемые MS-DOS, при старте системы, вы получите ситуацию, когда обращение к любой функции DOS приведет к вызову процедуры обработки «Ctrl-Break». При этом выполнение программы будет немного замедлено.

Процедура обработки «Ctrl-Break» дает возможность завершить программу в любой момент времени. Когда функция DOS распознает статус «Ctrl-Break», управление передается процедуре, на которую указывает вектор прерывания 23H. DOS применяет эту процедуру для завершения работающей программы. Но процедура

может быть переписана вами, с тем чтобы она удовлетворяла любым вашим требованиям. Эта процедура должна быть программируемой, чтобы перед завершением программы могли быть выполнены все критические операции. Может потребоваться выравнивание стека, для того чтобы SP указывал на второе слово от вершины (первое слово для программ COM) перед выполнением завершающей инструкции RET. Векторы прерывания, измененные программой, должны быть восстановлены, а все открытые устройства ввода/вывода - закрыты. Если были запрещены прерывания, то надо разрешить их. Все это должно обеспечить машине возможность нормально работать со следующей программой после завершения программы по «Ctrl-Break». Альтернатива - сделать процедуру обработки «Ctrl-Break», состоящей из одной инструкции IRET, что запрещает завершение программы таким способом.

Средний уровень

В данном примере выход из программы происходит после выравнивания стека. Процедура кончается инструкцией RET, а не IRET, поскольку в данном случае она действует точно так же, как и инструкция RET при нормальном завершении программы. В момент, когда она используется, указатель стека (SP) должен указывать на второе слово стека. Это предполагает, что программа имеет форму EXE. Помните, что стек помещает свое первое слово в самую старшую ячейку памяти, второе - в ячейку ниже и т.д. Если размер стека 400 байт, то надо установить SP на 396. Для программ COM надо устанавливать указатель стека на первое слово стека или просто завершать процедуру обработки «Ctrl-Break» прерыванием 20H.

```
;---это новая процедура обработки «Ctrl-Break»
C_B  PROC FAR
     MOV  AX,396        ;значение для второго слова стека
     MOV  SP,AX         ;выравниваем указатель стека
     RET                ;возврат в DOS
C_B  ENDP
;---изменение вектора прерывания
     PUSH DS            ;сохраняем регистр
     MOV  AX,SEG C_B    ;готовим адрес процедуры
     MOV  DS,AX
     MOV  DX,OFFSET C_B
     MOV  AH,25H        ;номер функции
     MOV  AL,23H        ;номер вектора
     INT  21H           ;изменяем вектор
     POP  DS            ;восстанавливаем регистр
```

Программа может в любое время проверить, был ли сделан запрос на выполнение процедуры обработки «Ctrl-Break». Надо поместить в AL 0 и вызвать функцию 33 прерывания 21H. При возврате DL будет содержать 1, если был установлен флаг прерывания по «Ctrl-Break», и 0 - в противном случае. Если при вызове поместить в AL 1, то статус будет установлен. В этом случае перед вызовом функции поместите в DL 0 или 1, чтобы флаг был установлен или очищен.

3.2.9. Перепрограммирование клавиши «PrtSc»

Клавиша «PrtSc» выдает звездочку (ASCII 42), если нажать ее одну, и расширенный код 114, если нажать ее вместе с клавишей «Ctrl». Но комбинация «Shift» + «PrtSc» имеет совершенно отдельный статус. Нажатие на другие клавиши заставляет прерывание клавиатуры помещать их коды в буфер клавиатуры (или, для клавиш-переключателей, записывать их состояние [3.1.7]). Нажатие клавиши не влияет на выполняемую программу, до тех пор пока программа не станет считывать символ клавиши из буфера клавиатуры. Но комбинация «Shift» + «PrtSc» заставляет прерывание клавиатуры немедленно передать управление процедуре, на которую указывает вектор прерывания 5. В некотором смысле она работает как аппаратное прерывание.

Прерывание 5 запрограммировано таким образом, чтобы выдать содержимое экрана на принтер. Но вектор прерывания может указывать на процедуру, предназначенную для совершенно другой цели. Например, изощренная программа имитации, которой требуются часы для завершения своей работы, может быть прервана в любое время комбинацией «Shift» + «PrtSc» для выдачи рапорта о текущем состоянии расчетов. Вы можете также захотеть, чтобы на принтер была послана копия графического экрана. «PrtSc» обеспечивает доступ к программе, которая находится резидентно в памяти во время загрузки MS-DOS [1.3.4]. Такая стратегия позволит вам написать утилиту, которая будет в состоянии работать из другого программного обеспечения.

Низкий уровень

Здесь приведена основная форма перепрограммирования процедуры. Не забудьте восстановить оригинальный вектор прерывания (F000:FF54) при завершении программы. Если вы забудете сделать это, то все пойдет нормально до тех пор, пока не будет нажата комбинация «Shift» + «PrtSc», а тогда произойдет сбой системы

(более полный пример программирования прерывания см. в [1.2.3]).

```
;---изменить вектор прерывания для «PrtSc»
        CLI                     ;запрет прерываний
        MOV  AX,SEG N_ROUT      ;получаем адрес процедуры
        MOV  DS,AX
        MOV  DX,OFFSET N_ROUT
        MOV  AL,5               ;номер изменяемого вектора
        MOV  AH,25H             ;номер функции
        INT  21H                ;изменяем вектор
        STI                     ;разрешаем прерывания

;---описание процедуры «PrtSc»
N_ROUT PROC FAR
        STI                     ;разрешаем прерывания
        PUSH AX                 ;сохраняем регистры

        MOV  CX,100             ;ваша процедура

        POP  AX                 ;восстанавливаем регистры
        IRET                    ;возврат из прерывания
N_ROUT ENDP
```

3.3. Сводка кодов клавиш и их назначение

Различные коды клавиш и коды символов могут приводить к недоразумениям. В приведенных ниже таблицах все они перечислены. Обратите внимание на следующие аномалии:

- клавиша «Ins» является единственной, которая при нажатии как выдает код символа в буфер клавиатуры, так и меняет статус регистра клавиш-переключателей;
- имеется 4 кода ASCII, которые могут быть получены двумя способами. ASCII 8 - нажатием клавиш «Backspace» и Ctrl-H, ASCII 9 - клавиш «Tab» и Ctrl-I, ASCII 13 - клавиш «Enter» и Ctrl-M, а ASCII 27 - клавиш «Esc» и Ctrl-[;
- символы, соответствующие 32 управляющим кодам ASCII , не выводятся на экран при использовании функций ввода с клавиатуры, обеспечивающих автоматическое эхо. Они могут быть

выведены либо с помощью функции 10H прерывания 10H, либо прямым выводом в память дисплея (оба способа обсуждаются в [4.3.1]);

- комбинации клавиши «Ctrl» с буквами алфавита генерируют однобайтовые коды ASCII. Все остальные комбинации «Ctrl» генерируют двухбайтовые (расширенные) коды;
- клавиша «5» дополнительной клавиатуры не действует, если установлен режим управления курсором клавишей «NumLock»;
- комбинации «Shift-PrtSc» и «Ctrl-Alt» (а также «SysReq» для AT) это те редкие случаи, когда комбинация клавиш приводит к немедленному вызову некоторой процедуры. Из них только первая перепрограммируема. Прерывание обработки «Ctrl-Break» (также перепрограммируемое) вызывается только тогда, когда статус «Ctrl-Break» обнаружен процедурой MS-DOS;
- любой код ASCII, кроме 0, может быть введен путем нажатия клавиши «Alt», набора кода ASCII на дополнительной клавиатуре и затем отпускания клавиши «Alt». Поскольку код 0 исключен, расширенные коды не могут быть введены таким способом.

Отметим, что вы практически ничего не можете сделать, чтобы преодолеть ограничения, накладываемые на недопустимые комбинации клавиш. Например, нельзя определить комбинацию Ctrl + CursorUp, принимая код «CursorUp», а затем проверяя регистр статуса переключателей для определения того, была ли нажата клавиша «Ctrl». Если «Ctrl» была нажата, то клавиша «CursorUp» вообще не выдает никакого кода.

3.3.1. Предопределенное использование клавиш

Относительно клавиш имеется ряд соглашений, которые должны выполняться всеми программами. Они приведены в техническом руководстве, и если программисты будут придерживаться их, то пользователю будет легко переходить от одной программы к другой. Заметим, однако, что программное обеспечение самой фирмы IBM не всегда следует этим соглашениям. Соглашения таковы:

ScrollLock	Переключает режим вывода на терминал, при котором перемещение курсора сдвигает экран, а не сам курсор.
CTRL 4/6	Сдвигает курсор на слово влево/вправо. Другая возможность: горизонтальный сдвиг экрана на позицию табуляции влево/вправо.
Pg Up	Возврат на 25 строк назад.
Pg Dn	Сдвиг на 25 строк вперед.
CTRL END	Удаление текста от позиции курсора до конца строки.

CTRL PgDn	Удаление текста от позиции курсора до конца экрана.
HOME	В тексте перемещает курсор к началу строки или документа. В меню возвращает в главное меню.
CTRL HOME	Чистит экран и помещает курсор в левый верхний угол.
END	Перемещает курсор к концу строки или к концу документа.
BACKSPACE/DELETE	DELETE уничтожает символ, на который указывает курсор, и сдвигает остаток строки на одну позицию влево. BACKSPACE удаляет символ слева от курсора и делает то же самое.
INS	Переключает режим вставки/замены.
TAB/BACKTAB	Перемещает курсор в следующую позицию табуляции: вправо - если была нажата одна «Tab», и влево - если вместе с клавишей Shift.
ESC	Выход из программы или процедуры.

3.3.2. Сводная таблица скан-кодов

Каждая клавиша генерирует два типа скан-кодов: "код нажатия", когда клавиша нажимается, и "код освобождения", когда клавиша отпускается. Для всех машин, кроме AT, код освобождения на 128 больше кода нажатия (бит 7 = 1). Таким образом, клавиша T создает код 20 при нажатии и код 148 при отпускании. AT использует одну и ту же цепочку битов для кодов нажатия и освобождения, но коды освобождения состоят из двух байтов, первый из которых всегда равен 0F0H. PCjr имеет специальный скан-код мнимой клавиши, номер 55. Этот код порождается, когда одновременно нажаты три или более клавиш, что помогает избежать ошибок при вводе. Прерывание клавиатуры отбрасывает этот код, и он не связывается ни с каким кодом ASCII или расширенным кодом.

Клавиши пишущей машинки

Клавиша	Код нажатия	Клавиша	Код нажатия	Клавиша	Код нажатия
"1"	2	"T"	20	"L"	38
"2"	3	"Y"	21	";"	39
"3"	4	"U"	22	"'"	40
"4"	5	"I"	23	"`"	41
"5"	6	"O"	24	"\"	43
"6"	7	"P"	25	"Z"	44

Клавиша	Код нажатия	Клавиша	Код нажатия	Клавиша	Код нажатия
"7"	8	"["	26	"X"	45
"8"	9	"]"	27	"C"	46
"9"	10	"A"	30	"V"	47
"0"	11	"S"	31	"B"	48
"-"	12	"D"	32	"N"	49
" = "	13	"F"	33	"M"	50
"Q"	16	"G"	34	","	51
"W"	17	"H"	35	"."	52
"E"	18	"J"	36	"/"	53
"R"	19	"K"	37	пробел	57

Управляющие клавиши

Esc	1	Ctrl	29	Alt	56
Backspace	14	left shift	42	CapsLock	58
Tab	15	right shift	54	NumLock	58
Enter	28	PrtSc	55	ScrollLock	70

Функциональные клавиши

F1	59	F5	63	F9	67
F2	60	F6	64	F10	68
F3	61	F7	65		
F4	62	F8	66		

Клавиши дополнительной клавиатуры

"7"	71	"5"	76	"3"	81
"8"	72	"6"	77	"0"	82
"9"	73	" + "	78	"."	83
"-"	74	"1"	79	Sys Req132 (только AT)	
"4"	75	"2"	80	мнимая55 (только PCjr)	

3.3.3. Сводная таблица кодов ASCII

Номера кодов от 0 до 31, управляющих кодов, обсуждаются более детально в [7.1.9]. Напоминаем, что любой код ASCII от 1 до 255 может быть введен с клавиатуры, если держать нажатой клавишу «Alt» при наборе номера кода на дополнительной клавиатуре (с соответственно установленным режимом «NumLock»). Когда клавиша «Alt» освобождается, код вводится.

Символ	Десятичное	Шестнадцатиричное	Двоичное
(null)	0	00	00000000
☺	1	01	00000001
☻	2	02	00000010
♥	3	03	00000011
♦	4	04	00000100
♣	5	05	00000101
♠	6	06	00000110
•	7	07	00000111
◘	8	08	00001000
○	9	09	00001001
◙	10	0A	00001010
♂	11	0B	00001011
♀	12	0C	00001100
♪	13	0D	00001101
♫	14	0E	00001110
☼	15	0F	00001111
►	16	10	00010000
◄	17	11	00010001
↕	18	12	00010010
‼	19	13	00010011
π	20	14	00010100
§	21	15	00010101
▬	22	16	00010110
↨	23	17	00010111
↑	24	18	00011000
↓	25	19	00011001
→	26	1A	00011010
←	27	1B	00011011
∟	28	1C	00011100
↔	29	1D	00011101
▲	30	1E	00011110
▼	31	1F	00011111
(space)	32	20	00100000
!	33	21	00100001
"	34	22	00100010
#	35	23	00100011
$	36	24	00100100
%	37	25	00100101
&	38	26	00100110

Символ	Десятичное	Шестнадцатиричное	Двоичное
'	39	27	00100111
(	40	28	00101000
)	41	29	00101001
*	42	2A	00101010
+	43	2B	00101011
,	44	2C	00101100
-	45	2D	00101101
.	46	2E	00101110
/	47	2F	00101111
0	48	30	00110000
1	49	31	00110001
2	50	32	00110010
3	51	33	00110011
4	52	34	00110100
5	53	35	00110101
6	54	36	00110110
7	55	37	00110111
8	56	38	00111000
9	57	39	00111001
:	58	3A	00111010
;	59	3B	00111011
<	60	3C	00111100
=	61	3D	00111101
>	62	3E	00111110
?	63	3F	00111111
@	64	40	01000000
A	65	41	01000001
B	66	42	01000010
C	67	43	01000011
D	68	44	01000100
E	69	45	01000101
F	70	46	01000110
G	71	47	01000111
H	72	48	01001000
I	73	49	01001001
J	74	4A	01001010
K	75	4B	01001011
L	76	4C	01001100
M	77	4D	01001101

Символ	Десятичное	Шестнадцатиричное	Двоичное
N	78	4E	01001110
O	79	4F	01001111
P	80	50	01010000
Q	81	51	01010001
R	82	52	01010010
S	83	53	01010011
T	84	54	01010100
U	85	55	01010101
V	86	56	01010110
W	87	57	01010111
X	88	58	01011000
Y	89	59	01011001
Z	90	5A	01011010
[	91	5B	01011011
\	92	5C	01011100
]	93	5D	01011101
^	94	5E	01011110
_	95	5F	01011111
`	96	60	01100000
a	97	61	01100001
b	98	62	01100010
c	99	63	01100011
d	100	64	01100100
e	101	65	01100101
f	102	66	01100110
g	103	67	01100111
h	104	68	01101000
i	105	69	01101001
j	106	6A	01101010
k	107	6B	01101011
l	108	6C	01101100
m	109	6D	01101101
n	110	6E	01101110
o	111	6F	01101111
p	112	70	01110000
q	113	71	01110001
r	114	72	01110010
s	115	73	01110011
t	116	74	01110100

Символ	Десятичное	Шестнадцатиричное	Двоичное
u	117	75	01110101
v	118	76	01110110
w	119	77	01110111
x	120	78	01111000
y	121	79	01111001
z	122	7A	01111010
{	123	7B	01111011
|	124	7C	01111100
}	125	7D	01111101
~	126	7E	01111110
⌂	127	7F	01111111
Ç	128	80	10000000
ü	129	81	10000001
é	130	82	10000010
â	131	83	10000011
ä	132	84	10000100
à	133	85	10000101
å	134	86	10000110
ç	135	87	10000111
ê	136	88	10001000
ë	137	89	10001001
è	138	8A	10001010
ï	139	8B	10001011
î	140	8C	10001100
ì	141	8D	10001101
Ä	142	8E	10001110
Å	143	8F	10001111
É	144	90	10010000
æ	145	91	10010001
Æ	146	92	10010010
ô	147	93	10010011
ö	148	94	10010100
ò	149	95	10010101
û	150	96	10010110
ù	151	97	10010111
ÿ	152	98	10011000

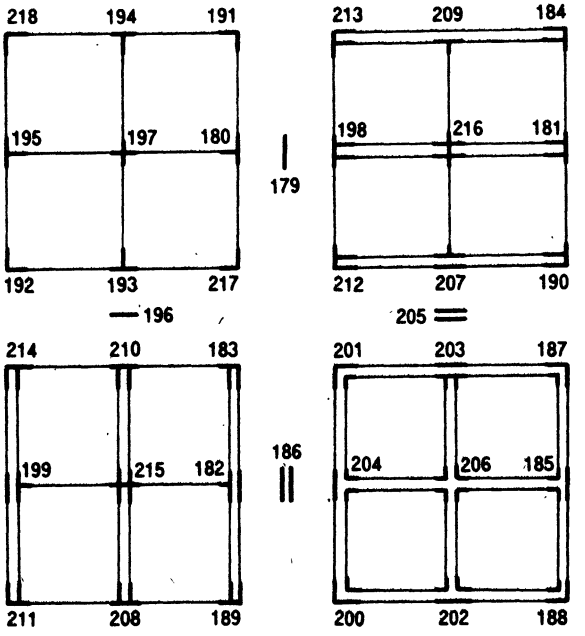
Символ	Десятичное	Шестнадцатиричное	Двоичное
Ö	153	99	10011001
Ü	154	9A	10011010
¢	155	9B	10011011
£	156	9C	10011100
¥	157	9D	10011101
Pt	158	9E	10011110
f	159	9F	10011111
á	160	A0	10100000
í	161	A1	10100001
ó	162	A2	10100010
ú	163	A3	10100011
ñ	164	A4	10100100
Ñ	165	A5	10100101
ª	166	A6	10100110
º	167	A7	10100111
¿	168	A8	10101000
⌐	169	A9	10101001
¬	170	AA	10101010
½	171	AB	10101011
¼	172	AC	10101100
¡	173	AD	10101101
«	174	AE	10101110
»	175	AF	10101111
	176	B0	10110000
▒	177	B1	10110001
▓	178	B2	10110010
│	179	B3	10110011
┤	180	B4	10110100
╡	181	B5	10110101
╢	182	B6	10110110
╖	183	B7	10110111
╕	184	B8	10111000
╣	185	B9	10111001
║	186	BA	10111010
╗	187	BB	10111011
╝	188	BC	10111100
╜	189	BD	10111101
╛	190	BE	10111110
┐	191	BF	10111111

Символ	Десятичное	Шестнадцатиричное	Двоичное
└	192	C0	11000000
┴	193	C1	11000001
┬	194	C2	11000010
├	195	C3	11000011
─	196	C4	11000100
┼	197	C5	11000101
╞	198	C6	11000110
╟	199	C7	11001000
╚	200	C8	11001001
╔	201	C9	11001010
╩	202	CA	11001011
╦	203	CB	11001100
╠	204	CC	11001101
═	205	CD	11001110
╬	206	CE	11001111
╧	207	CF	11010000
╨	208	D0	11010001
╤	209	D1	11010010
╥	210	D2	11010011
╙	211	D3	11010100
╘	212	D4	11010101
╒	213	D5	11010110
╓	214	D6	11010111
╫	215	D7	11011000
╪	216	D8	11011001
┘	217	D9	11011010
┌	218	DA	11011011
▀	219	DB	11011100
▄	220	DC	11011101
▌	221	DD	11011110
▐	222	DE	11011111
▀	223	DF	11100000
α	224	E0	11100001
β	225	E1	11100010
Γ	226	E2	11100011
π	227	E3	11100100
Σ	228	E4	11100101
σ	229	E5	11100110
μ	230	E6	

Символ	Десятичное	Шестнадцатиричное	Двоичное
τ	231	E7	11100111
Φ	232	E8	11101000
ϕ	233	E9	11101001
Ω	234	EA	11101010
δ	235	EB	11101011
∞	236	EC	11101100
∅	237	ED	11101101
∈	238	EE	11101110
∩	239	EF	11101111
≡	240	F0	11110000
±	241	F1	11110001
≥	242	F2	11110010
≤	243	F3	11110011
⌠	244	F4	11110100
⌡	245	F5	11110101
÷	246	F6	11110110
≈	247	F7	11110111
°	248	F8	11111000
•	249	F9	11111001
·	250	FA	11111010
√	251	FB	11111011
n	252	FC	11111100
2	253	FD	11111101
■	254	FE	11111110
(blank 'FF')	255	FF	11111111

3.3.4. Сводка кодов псевдографики для построения рамок

Ниже приведены номера кодов ASCII для символов псевдографики, используемых при построении линий и рамок.



3.3.5. Сводная таблица расширенных кодов

Значение 2-го байта	Соответствующие клавиши
15	Shift + Tab ("back-tab")
16-25	Alt + Q - Alt + P (верхний ряд букв)
30-38	Alt + A - Alt + L (средний ряд букв)
44-50	Alt + Z - Alt + M (нижний ряд букв)
59-68	Функциональные клавиши F1 - F10
71	Home
72	Cursor-up (стрелка вверх)
73	PgUp
75	Cursor-left (стрелка влево)
77	Cursor-right (стрелка вправо)
79	End
80	Cursor-down (стрелка вниз)
81	PgDn
82	Ins
83	Del
84-93	F1-F10 + Shift
94-103	F1-F10 + Ctrl
104-113	F1-F10 + Alt
114	Ctrl + PrtSc
115	Ctrl + Cursor-left
116	Ctrl + Cursor-right
117	Ctrl + End
118	Ctrl + PgDn
119	Ctrl + Home
120-131	Alt + 1 - Alt + = (верхний ряд)
132	Ctrl + PgUp

Глава 4. Вывод на терминал

4.1. Управление выводом на терминал

В этой главе рассмотрены монохромный адаптер, цветной графический адаптер, видеосистема PCjr и улучшенный графический адаптер (EGA). Все 4 системы базируются на микросхеме Motorola 6845 CRTC (cathode ray tube controller); хотя EGA на самом деле использует заказную микросхему, основанную на принципах микросхемы 6845. Эта микросхема выполняет массу технических задач, которые обычно не интересуют программиста. Однако она также устанавливает режим экрана, управляет курсором и (для цветного графического адаптера) цветом. Микросхема легко программируется напрямую, хотя процедуры операционной системы позволяют управлять большинством ее действий. PCjr имеет вспомогательную микросхему для дисплея "video gate array" (массив ворот дисплея), которая рассматривается в этом разделе вместе с микросхемой 6845. Архитектура EGA отличается от всех остальных адаптеров, поэтому EGA обсуждается отдельно. Среди не-EGA систем существует совместимость по использованию адресов портов, но есть и существенные различия. Некоторые адреса портов EGA такие же, как и у других систем. Все видеосистемы используют буфера, в которые отображаются данные для изображения на экране. Экран периодически обновляется сканированием этих данных. Размер и расположение буферов меняется с системой, а также режимом экрана, количеством заранее отведенной памяти. Когда в буфере хранится несколько образов экрана, каждый отдельный образ называют дисплейной страницей.

Монохромный адаптер. Монохромный адаптер имеет 4K байт памяти на плате, начиная с адреса B0000H (т.е. B000:0000). Этой памяти хватает только для хранения одной 80-символьной страницы текста.

Цветной графический адаптер. Цветной графический адаптер имеет 16K байт памяти на плате, начиная с адреса памяти B8000H. Этого достаточно для отображения одного графического экрана, без страниц, или для отображения от четырех до восьми экранов текста в зависимости от числа символов в строке - 40 или 80.

PCjr. PCjr имеет видеосистему, которая на самом деле является улучшенной версией цветного графического адаптера. Она уникальна тем, что использует для видеобуфера обычную оперативную память системы. Когда BIOS инициализирует систему, верхние 16K установленной памяти отводятся под буфер терминала. Таким образом, адрес буфера зависит от того, сколько памяти в системе. Для добавочных дисплейных страниц могут быть отведены блоки памяти в других местах, а также начальный объем может быть уменьшен до 4K и поддерживать только один экран текста.

EGA. EGA может быть снабжен 64K, 128K или 256K памяти. Кроме использования в качестве видеобуфера, эта память может также хранить битовые описания вплоть до 1024 символов (как объяснено в [4.3.4]). Стартовый адрес буфера дисплея программируемый, поэтому буфер начинается с адреса A000H для улучшенных графических режимов и с B000H и B800H для совместимости со стандартными монохромным и цветным графическим режимами. В большинстве случаев EGA занимает два сегмента с адресами от A000H до BFFFH, даже когда имеется 256K памяти. Это возможно, поскольку в некоторых режимах два или более байтов памяти дисплея считываются из одних и тех же адресов. Доступное число страниц зависит как от режима экрана, так и от количества имеющейся памяти. Вследствие своей сложности EGA имеет ПЗУ на 16K байт, которое заменяет и расширяет процедуры работы с терминалом BIOS. Начало области ПЗУ - адрес C000:0000.

В текстовых режимах буфера начинаются с данных для верхней строки экрана от левого угла. Дальнейшие данные переносятся с правого конца одной строки на левый конец следующей, как будто экран представляется одной большой строкой - с точки зрения видеобуфера так оно и есть. Однако в графических режимах буфер может быть разделен на 2 или 4 части. У цветного графического адаптера и PCjr различные части буфера содержат информацию, относящуюся к каждой второй или каждой четвертой линии точек на экране. У EGA каждая часть буфера

содержит один бит из двух или четырех, которые определяют цвет данной точки экрана.

При выводе текста различные видеосистемы работают одинаково. Для экрана отводится 4000 байт, так что на каждую из 2000 позиций экрана приходится 2 байта (25 строк x 80 символов). Первый байт содержит код ASCII. Аппаратура дисплея преобразует номер кода ASCII в связанный с ним символ и посылает его на экран. Второй байт (байт атрибутов) содержит информацию о том, как должен быть выведен данный символ. Для монохромного дисплея он устанавливает, будет ли данный символ подчеркнут, выделен яркостью или негативом, либо применяется комбинация этих атрибутов. В цветовых системах байт атрибутов устанавливает основной и фоновый цвета символа. В любом случае ваша программа может писать данные прямо в буфер терминала, что значительно повышает скорость вывода на экран.

Все системы, кроме монохромной, предоставляют набор цветных графических режимов, которые отличаются как разрешением, так и числом одновременно выводимых цветов. И PCjr, и EGA могут одновременно выводить 16 цветов, причем EGA может выбирать эти 16 из набора 64 цветов. При использовании 16 цветов каждая точка экрана требует четырех бит памяти, поскольку 4 бита могут хранить числа от 0 до 15. По аналогии четырехцветная графика требует только 2 бита на точку. При двухцветной графике представление восьми точек может быть упаковано в один байт видеобуфера. Количество памяти, требуемое для данного режима экрана, легко вычислить, если известно количество выводимых в этом режиме точек и количество битов, необходимое для описания одной точки. Текст легко комбинируется с графикой (BIOS рисует символы на графическом экране), и вы можете создавать свои специальные символы.

4.1.1. Программирование контроллера дисплея 6845

Все видеосистемы строятся вокруг микросхемы контроллера видеотерминала Motorola 6845 (EGA использует заказную микросхему, основанную на микросхеме 6845). Применение микросхемы во многом идентично в адаптерах: монохромном, цветном и PCjr; но EGA сильно отличается от других адаптеров и по этой причине мы рекомендуем избегать прямого программирования микросхемы, когда BIOS может выполнить работу за вас. Говоря общими словами, микросхема 6845 устанавливает видеодисплей в один из алфавитно-цифровых или графических режимов. Она выполняет основную работу по интерпретации номеров кодов ASCII и поиску данных для вывода соответствующих символов в микросхеме ПЗУ

(а иногда в оперативной памяти), декодирует значения атрибутов цвета и соответственно устанавливает экран, а также создает курсор и управляет им. В архитектуре EGA часть этих функций распределена между другими микросхемами.

Микросхема 6845 имеет 18 управляющих регистров, пронумерованных от 0 до 17. Первые 10 регистров фиксируют горизонтальные и вертикальные параметры дисплея. Эти регистры, как правило, неинтересны для программистов, поскольку они автоматически устанавливаются BIOS при изменении режима экрана. Не советуем экспериментировать с этими регистрами, поскольку можно испортить терминал. Регистры имеют размер 8 бит, но некоторые связаны в пары, чтобы хранить 16-битовые величины. Пары #10-11 и #14-15 устанавливают форму [4.2.4] и местоположение [4.2.1] курсора. Пара #12-13 управляет страницами дисплея [4.5.3]. Пара #16-17 сообщает позицию светового пера [7.3.2]. Большинство регистров доступно только для записи; регистр адреса курсора можно и читать, и писать, а регистр светового пера предназначен только для чтения. EGA имеет 6 добавочных регистров, которые связаны с техническими деталями. Регистр 20 наиболее интересен; он определяет, какая линия сканирования в строке символа используется для подчеркивания.

Доступ ко всем 18 регистрам осуществляется через один и тот же порт, адрес которого для монохромного адаптера - 3B5H, для цветного адаптера и PCjr - 3D5H (заметим, что все адреса портов для монохромного адаптера такие же, как и для цветного, за исключением того, что средней цифрой является B, а не D). EGA использует один из этих двух адресов в зависимости от того, присоединен ли к нему цветной или монохромный монитор. Для записи в регистр монохромного адаптера надо сначала в регистр адреса, расположенный в порте 3B4H (3D4H для цветного), послать номер требуемого регистра. Тогда следующий байт, посланный в порт с адресом 3B5H, будет записан в этот регистр. Поскольку регистры, интересные для программиста, используются попарно, надо сначала записать в адресный регистр, потом в первый регистр пары, потом снова в адресный регистр и, наконец, во второй регистр пары. Адреса портов смежные, поэтому легче всего адресовать их, используя инструкции INC и DEC, как в следующем примере:

```
;---запись в регистры 11 и 12 микросхемы 6845 (данные в BX)
;---выбираем регистр младшего байта
MOV  DX,3B4H          ;порт адресного регистра
MOV  AL,11            ;номер регистра для младшего байта
OUT  DX,AL            ;посылаем номер регистра
```

```
;---посылаем байт
INC   DX                ;увеличиваем адрес порта
MOV   AL,BL             ;берем младший байт
OUT   DX,AL             ;посылаем его в регистр 11
;---выбираем регистр старшего байта
DEC   DX                ;восстанавливаем адрес порта
MOV   AL,12             ;номер регистра для старшего байта
OUT   DX,AL             ;посылаем номер регистра
;---посылаем байт
INC   DX                ;увеличиваем адрес порта
MOV   AL,BH             ;берем старший байт
OUT   DX,AL             ;посылаем его в регистр 12
```

У монохромного и цветного адаптеров имеются еще три порта, которые важны для программистов. Они имеют адреса 3B8H, 3B9H и 3BAH для монохромного и 3D8H, 3D9H и 3DAH - для цветного адаптера. Первый устанавливает режим экрана, второй связан в основном с установкой цветов экрана, а третий сообщает полезную информацию о статусе дисплея.

PCjr использует не все эти адреса аналогичным образом. Вместо этого он держит часть информации, относящейся к портам, в микросхеме массива ворот дисплея, основное назначение которой обеспечить дополнительное управление цветами экрана. Доступ к массиву ворот дисплея осуществляется через порт с адресом 3DAH. У цветного адаптера этот порт возвращает байт статуса; у PCjr этот порт также возвращает байт статуса при использовании инструкции IN, но он предоставляет доступ к массиву ворот, когда используется инструкция OUT. Массив ворот дисплея имеет следующие регистры:

Номер	Назначение
0	режим управления 1
1	маска набора цветов (палитры)
2	цвет границы
3	режим управления 2
4	сброс
10H-1FH	назначение цветов палитры

Доступ ко всем регистрам осуществляется через порт 3DAH. Сначала надо послать в этот порт номер требуемого регистра, а затем - значение регистра. Порт автоматически переключается между функциями работы с адресами и с данными. Чтобы он ожи-

дал ввод адреса, надо прочитать из него. Отдельные регистры обсуждаются по мере изложения материала этой главы.

Особый интерес представляют 16 регистров палитры с номерами от 10H до 1FH. Каждый регистр имеет размер всего 4 бита, этого достаточно, чтобы хранить 16 кодовых номеров для 16 возможных цветов. Для каждой позиции символа или точки на экране видеобуфер содержит данные, указывающие, каким цветом должен выводиться объект. Такую информацию называют данными атрибутов. В отличие от цветного графического адаптера PCjr не использует данные атрибутов для непосредственного определения цвета, который будет выводиться. Вместо этого данные атрибутов являются указателями на один из 16 регистров палитры, а число, содержащееся в регистре, определяет, каким цветом будет выводиться данный символ. При таком методе программе нужно изменить только установку регистра палитры, и все символы или точки с соответствующим атрибутом изменят свой цвет. Регистры палитры работают во всех режимах как текстовых, так и графических.

EGA распределяет эти функции между микросхемой контроллера атрибутов (адрес порта 3C0H) и двумя микросхемами контроллера графики (адреса портов 3CCH-3CFH). Контроллер атрибутов содержит 16 регистров палитры EGA, пронумерованных от 00 до 0FH. Эти регистры могут содержать 6-битовые коды цветов, когда EGA связан с улучшенным цветным дисплеем, поэтому могут быть использованы любые 16 цветов из набора 64. В [4.4.1] показано, как программировать регистры палитры для PCjr и EGA.

4.1.2. Установка/проверка режима дисплея

Монохромный адаптер поддерживает один режим терминала, цветной графический - семь, PCjr - десять, а EGA - двенадцать. Система PCjr более гибкая, чем монохромный или цветной адаптеры, поскольку она предоставляет широкий выбор цветов в режимах с двумя и четырьмя цветами, а также серые тени в черно-белом режиме. EGA еще более сложен, он поддерживает палитру из 64 цветов, графику на монохромном дисплее и вывод в 43 строки. Ниже приведен перечень различных режимов:

Номер	Режим	Адаптеры
0	40*25(320*200) B&W алфавитно-цифровой	цветной, PCjr, EGA
1	40*25 (320*200) цветной алфавитно-цифровой	цветной, PCjr, EGA
2	80*25 (640*200) B&W алфавитно-цифровой	цветной, PCjr, EGA
3	80*25 (640*200) цветной алфавитно-цифровой	цветной, PCjr, EGA

4	320*200 4-цветная графика	цветной, PCjr, EGA
5	320*200 B&W графика (4 тени на PCjr)	цветной, PCjr, EGA
6	640*200 B&W графика	цветной, PCjr, EGA
7	80*25 (720*350) B&W алфавитно-цифровой	монохромный, EGA
8	160*200 16-цветная графика	PCjr
9	320*200 16-цветная графика	PCjr
A	640*2004-цветная графика	PCjr
B	зарезервирован для EGA	
C	зарезервирован для EGA	
D	320*200 16-цветная графика	EGA
E	640*200 16-цветная графика	EGA
F	640*350 4-цветная графика на монохромном	EGA
10	640*350 4- или 16-цветная графика	EGA

EGA разрешает иметь 8 страниц в режиме 7 - стандартном монохромном текстовом режиме. Режимы 0-6 полностью совместимы и используют память одинаковым образом. При условии, что переключатели на EGA установлены для работы с улучшенным цветным дисплеем фирмы IBM, традиционные текстовые режимы выводятся с высоким разрешением, а рисунок символов, состоит из 8*14 точек, а не из 8*8, как обычно.

BIOS хранит однобайтовую переменную по адресу 0040:0049, в которой содержится номер текущего режима. Байт по адресу 0040:004A дает число символов в строке в текстовом режиме.

Высокий уровень

Бейсик использует операторы SCREEN и WIDTH для управления режимом экрана. PCjr применяет эти операторы несколько другим способом, чем монохромный и цветной адаптеры, и это будет обсуждаться ниже. Один оператор SCREEN устанавливает режим для цветного адаптера. За оператором стоит номер кода, устанавливающий разрешение, где:

0	текстовый режим
1	графический режим среднего разрешения
2	графический режим высокого разрешения

SCREEN 1 устанавливает графический режим среднего разрешения. Второй параметр включает и выключает цвет. Этот параметр не имеет смысла для режима высокого разрешения на цвет-

ном адаптере, поскольку разрешен только черно-белый режим. Для текстовых режимов 0 в качестве второго параметра выключает цвет, а 1 включает. Оператор SCREEN 0,0 устанавливает текстовый черно-белый режим. Для графического режима ситуация обратная: 0 включает цвет, а 1 выключает. Поэтому оператор SCREEN 1,1 устанавливает черно-белый графический режим среднего разрешения.

Все режимы первоначально выводятся черно-белыми. Чтобы закрасить экран фоновым цветом, должен быть использован оператор COLOR (см. [4.1.3]). В графическом режиме одного оператора COLOR достаточно, чтобы изменить весь фон на указанный цвет. Но для текстового режима вы должны после оператора COLOR перейти к оператору CLS.

В текстовых режимах в строке может быть 40 или 80 символов. Для установки требуемого числа символов в строке нужен оператор WIDTH. WIDTH 40 дает 40 символов в строке, а WIDTH 80 - 80. Другие значения недопустимы. Если оператор WIDTH используется в графическом режиме (SCREEN 1 или SCREEN 2), то WIDTH 40 переводит экран в режим среднего разрешения, а WIDTH 80 - в режим высокого разрешения. Вот несколько примеров:

```
100 SCREEN 0,1: WIDTH 40  'цветной текстовый режим с 40 символами
100 SCREEN 0,0: WIDTH 80  'цветной дисплей, в монохромном режиме
100 SCREEN 1,0:           цветная графика среднего разрешения

500 WIDTH 80              'переводим в режим высокого разрешения
```

Монохромный монитор может быть переведен в режим 40 символов в строке операторами SCREEN 0: WIDTH 40. Для восстановления режима с 80 символами введите WIDTH 80. В режиме с 40 символами они сохраняют свою обычную ширину, поэтому будет использоваться только левая часть экрана. Строка переносится после 40-го столбца и невозможно поместить курсор в правую половину экрана с помощью оператора LOCATE. CLS чистит только левую часть экрана. Трудно представить программу, которая использовала бы это свойство, но оно действительно позволяет программе принимать ввод (скажем, через оператор INPUT), в то время как пользователь продолжает печатать в левой половине экрана, оставляя правую половину экрана для возможной корректировки вводимой информации. При этом любой вывод в правую половину экрана возможен только при прямом обращении к памяти дисплея, как объяснено в [4.3.1].

В Бейсике PCjr применяется 7 номеров режимов:

Номер	Режим
0	текстовый режим, ширина может быть 40 или 80
1	4-цветная графика среднего разрешения
2	2-цветная графика высокого разрешения
3	16-цветная графика низкого разрешения
4	4-цветный режим среднего разрешения
5	16-цветный режим среднего разрешения
6	4-цветный режим высокого разрешения

Последние четыре режима требуют дискеты с Бейсиком. Размер страницы определяет количество памяти для одного экрана (дисплейные страницы обсуждаются в [4.5.3]). Программа должна отвести соответствующее количество памяти перед установкой режима. Это делается оператором CLEAR. За оператором CLEAR должны следовать три числа, определяющие отводимую память, третье из этих чисел устанавливает размер видеобуфера (первые два параметра обсуждаются в [1.3.1]). Например, размер для видеобуфера 16K, устанавливаемый по умолчанию, выделяется командой CLEAR ,,16384. К сожалению, размер видеобуфера указывается в байтах, поэтому он не равен круглому числу типа 4000 или 32000, а равен 4096 или 32768. Помните, что 2K = 2^11, 4K = 2^12, 16K = 2^14, а 32K = 2^15. Для выделения трех страниц по 16K введите CLEAR ,,3 x 2^14. Этот оператор должен помещаться в самом начале программы, поскольку при использовании оператора CLEAR все переменные очищаются. Отметим также, что при создании нескольких страниц страница 0 начинается с младших адресов памяти.

К моменту выхода этой книги Бейсик не поддерживает дополнительные режимы терминала EGA. В [4.3.3] приведена подпрограмма на машинном языке, которая позволит вам установить эти режимы.

Средний уровень

Функция 0 прерывания 10H устанавливает режим дисплея. В AL должен находиться номер режима от 0 до A. Чтобы установить цветной графический режим среднего разрешения, надо:

```
MOV  AH,0      ;номер функции
MOV  AL,4      ;номер требуемого режима
INT  10H       ;устанавливаем режим
```

Для определения текущего графического режима применяют функцию F прерывания 10H. Прерывание возвращает номер режима в AL. Оно также дает номер текущей страницы дисплея в BH и число символов в строке в AH.

```
MOV  AH,0FH              ;номер функции
INT  10H                 ;получение информации о режиме дисплея
MOV  MODE_NUMBER,AL      ;номер режима в AL
MOV  NUMBER_COLS,AH      ;число символов в строке в AH
MOV  CURRENT_PAGE,BH     ;номер текущей страницы в BH
```

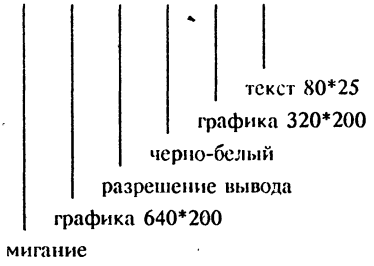
MS-DOS обеспечивает также Esc-последовательности для установки и сброса режимов дисплея. Для этого необходимо предварительно загрузить драйвер ANSI.SYS, как объяснено в приложении Д. Управляющая строка имеет вид ESC [= #h, где # - номер режима, указанный как код ASCII, а ESC обозначает один символ с кодом ASCII 27. Например:

```
;---в сегменте данных
M_RES_COL      DB   27, '[=4h$'
MED_RES_B&W    DB   27, '[=5h$'
;---установка цветного графического режима среднего разрешения
               MOV  AH,9            ;номер функции вывода строки
               LEA  DX,M_RES_COL    ;DS:DX должны указывать на строку
               INT  21H             ;изменение режима
```

Низкий уровень

В данном случае цветной адаптер, монохромный адаптер и PCjr рассматриваются отдельно, поскольку они существенно различаются. Цветной графический адаптер имеет регистр, который устанавливает режим дисплея. Он расположен в порте с адресом 3D8H. Биты 0, 1, 2 и 4 хранят установку. Бит 0 устанавливает 40 символов в строке, когда он равен 0, и 80, когда он равен 1. Бит 1 устанавливает дисплей в текстовый режим, когда он равен 0, и в графический, когда он равен 1. Бит 2 устанавливает цветной режим, когда он равен 0, и черно-белый, когда он равен 1. И наконец, бит 4 устанавливает для графического режима среднее разрешение, когда он равен 0, и высокое разрешение, когда он равен 1 (бит 2 должен быть равен 1). Ниже приведены возможные комбинации:

Режим	Биты: 5	4	3	2	1	0
0. 40*25, черно-белый, текст	1	0	1	1	0	0
1. 40*25, цветной, текст .	1	0	1	0	0	0
2. 80*25, черно-белый, текст	1	0	1	1	0	1
3. 80*25, цветной, текст	1	0	1	0	0	1
4. 320*200, черно-белый, графика	0	0	1	1	1	0
5. 320*200, цветной, графика	0	0	1	0	1	0
6. 640*200, черно-белый, графика	0	1	1	1	1	0



Изменение этих битов не приводит к изменению режима дисплея. Нужно еще много шагов, включающих изменение параметров первых 10 регистров по адресу порта 3D5H. BIOS заботится обо всем этом, и не имеет смысла заниматься подобной деятельностью. Однако иногда целесообразно реинициализировать регистр режима в его текущем режиме, изменяя биты 3 и 5, которые на самом деле не отвечают за установку режима. Когда бит 5 сброшен в 0, то он запрещает атрибут мигания символов; в этом случае, если старший бит байта атрибутов установлен, то это приводит к выводу фонового цвета высокой интенсивностью (см. пример в [4.1.3]). Бит 3 этого регистра управляет разрешением вывода. Когда он равен 0, весь экран закрашивается в цвет рамки, но видеобуфер не очищается. Вывод мгновенно возвращается, когда значение этого бита меняется на 1. Это свойство позволяет избежать интерференции экрана при сдвигах [4.5.1]. Некоторые утилиты используют его для того, чтобы зря не утомлять фосфорное покрытие трубки терминала, когда компьютер включен, но не используется. Отметим также, что два старших бита регистра не принимаются во внимание.

Соответствующий адрес порта монохромного адаптера - 3B8H. Имеют значение только три бита. Бит 0 устанавливает высокое разрешение, которое является единственным допустимым режимом для монохромного дисплея. Если этот бит равен 0, то компьютер перестает работать. Два других значащих бита - это биты 3 и 5,

которые управляют разрешением вывода и миганием в точности так же, как и для цветного адаптера.

PCjr распределяет информацию, содержащуюся в одном порте для монохромного и цветного адаптеров. Массив ворот дисплея имеет два регистра режима, номер 0 и 3. Для доступа к этим регистрам надо послать номер регистра в порт с адресом 3DAH, а затем записать данные по тому же адресу (чтение этого порта обеспечивает восприятие первой записи в него как указание номера требуемого регистра). Вот значение битов этих регистров:

Регистр 0:

бит	0	1 = текст, 80*25 и режимы 5 и 6, иначе 0
	1	1 = графический режим, 0 = текстовый
	2	1 = запрет цветов, 0 = разрешение цветов
	3	1 = разрешение вывода, 0 = запрет вывода
	4	1 = 16-цветный режим, 0 = все остальные режимы

Регистр 3:

бит	0	всегда 0
	1	1 = разрешение мигания, 0 = 16 фоновых цветов
	2	всегда 0
	3	1 = 2-цветная графика, 0 = все остальные режимы

Как и в двух предыдущих случаях, не стоит устанавливать регистры прямо из программы, так как нужно еще много работы для программирования микросхемы 6845. Но каждый из двух регистров содержит бит, который иногда приходится программно модифицировать, а поскольку эти регистры только для записи, вам необходимо понимать значение всех их битов: бита разрешения вывода в регистре 0 и бита разрешения мигания в регистре 3. Их действие было описано ранее, и возможное их использование еще не раз будет обсуждаться в этой главе (см. [4.5.1] и [4.1.3]).

EGA имеет два регистра, управляющих режимом дисплея. Адрес порта первого - 3D5H. Этот регистр не содержит ни одного бита, связанного с чем-либо другим, поэтому нет никаких причин обращаться к нему. Второй регистр имеет адрес порта 3C0H и содержит бит, который выбирает, будет ли бит 7 байта атрибутов соответствовать миганию или высокой интенсивности. Этот вопрос обсуждается в [4.1.3].

4.1.3. Установка атрибутов/цветов символов

Когда дисплей установлен в текстовый режим в любой из видеосистем, каждой позиции символа на экране отводится два байта памяти. Первый байт содержит номер кода ASCII символа, а второй - атрибуты символа. Цветной адаптер и PCjr могут выводить в цвете как сам символ, так и всю область, отведенную данному символу (фоновый цвет). Монохромный адаптер ограничен только черным и белым цветом, но он может генерировать подчеркнутые символы, чего не могут делать цветной адаптер и PCjr. Все три системы могут выдавать мигающие символы и негативное изображение. Они могут также создавать символы с высокой интенсивностью, хотя для цветного адаптера и PCjr повышенная интенсивность символа на самом деле приводит к другому цвету (восемь основных цветов имеют версии с повышенной интенсивностью, что дает набор 16 цветов). EGA может делать все, что и остальные системы, а также многое другое. В частности, на улучшенном дисплее он выводит подчеркнутые цветные символы, поскольку матрица изображения символов 8*14 предоставляет такую возможность.

Атрибуты цвета

Для указания цветов экрана одни и те же номера кодов используются в Бейсике и прерываниями операционной системы:

0 - черный	8 - серый
1 - синий	9 - голубой
2 - зеленый	10 - светло-зеленый
3 - циан*	11 - светлый циан
4 - красный	12 - светло-красный
5 - магента*	13 - светлая магента
6 - коричневый	14 - желтый
7 - белый	15 - ярко-белый

Младшие четыре бита байта атрибутов устанавливают цвет самого символа (бит 3 включает высокую интенсивность). Следующие три бита устанавливают фон символа. И при обычных обстоятельствах старший бит включает и выключает мигание. Таким образом:

---

*Цвет магента (magenta) приблизительно соответствует фиолетовому цвету, а циан (cyan) - голубому. - *Примеч. пер.*

когда бит	0 = 1,	синий включается в основной цвет
	1 = 1,	зеленый включается в основной цвет
	2 = 1,	красный включается в основной цвет
	3 = 1,	символ выводится с высокой интенсивностью
	4 = 1,	синий включается в фоновый цвет
	5 = 1,	зеленый включается в фоновый цвет
	6 = 1,	красный включается в фоновый цвет
	7 = 1,	символы мигают

Биты 0-2 и 4-6 содержат одни и те же компоненты цветов для самих символов и фона. Эти трехбитовые группы позволяют восемь возможных комбинаций. Когда включается бит высокой интенсивности, добавляются еще восемь цветов. Шестнадцать возможных цветов получаются из этих установок битов следующим образом:

			Интенсивность	
Красный	Зеленый	Синий	Низкая	Высокая
0	0	0	черный	серый
0	0	1	синий	светло-синий
0	1	0	зеленый	светло-зеленый
0	1	1	циан	светлый циан
1	0	0	красный	светло-красный
1	0	1	магента	светлая магента
1	1	0	коричневый	желтый
1	1	1	белый	ярко-белый

Можно иметь 16 цветов и для фонового цвета. В этом случае бит 7 должен служить указателем высокой интенсивности для фона, а не указателем мигания символов. Для цветного адаптера надо изменить бит 5 порта с адресом 3D8H в 0, как показано ниже. Поскольку этот порт доступен только для записи, то все остальные биты должны быть переустановлены. Эта возможность доступна только в двух случаях: текстовых режимов с 40 и с 80 символами в строке. Для режима с 80 символами надо послать в порт число 9, а для режима с 40 символами - число 8. Чтобы вернуть мигание, надо добавить к обоим значениям 32. Для PCjr надо сбросить в 0 бит 1 регистра 3 массива ворот дисплея. Все остальные биты должны быть равны нулю, кроме номера 3, который должен быть установлен для режима двухцветной графики. Кроме этого режима, для установки бита мигания надо сначала прочитать

порт с адресом 3DAH, чтобы подготовить массив ворот дисплея, затем послать в него 3, чтобы указать регистр, и затем послать 0, чтобы установить бит мигания. При завершении программы всегда надо восстанавливать мигание, так как следующая программа может полагаться на это.

EGA также может разрешать/запрещать мигание, хотя в этом случае адрес порта - 3C0H. Сначала надо прочитать порт 3DAH, чтобы получить доступ к адресному регистру в 3C0H. Затем надо послать в 3C0H 10H, чтобы указать соответствующий регистр. Наконец, надо послать данные по тому же адресу. Поскольку этот регистр только для записи, все биты должны быть правильно установлены. Мигание включается установкой бита 3, а выключается сбросом этого бита. Все остальные биты в цветном текстовом режиме должны быть равны 0.

Для цветного адаптера, когда символы выводятся на дисплей в цветном графическом режиме, они изображаются в текущем фоновом цвете. Операторы, которые выводят на экран, как в Бейсике, так и в MS-DOS (прерывание 21H) ограничены выводом символов в третьем цвете используемой палитры (имеются две палитры из трех цветов, см. [4.4.1]). В палитре 0 символы желтые/коричневые, а в палитре 1 они белые. Процедуры вывода символов BIOS (прерывание 10H), однако, могут указать любой из трех цветов палитры. С другой стороны, для PCjr цвет, назначенный определенной позиции палитры, может быть изменен, поэтому для вывода символов могут быть использованы любые цвета.

Для PCjr цвета, соответствующие данным кодовым номерам, могут быть изменены. Каждый кодовый номер связан с регистром палитры в массиве ворот дисплея [4.1.1]. Эти регистры пронумерованы от 10H до 1FH, что соответствует кодам от 0 до 15. Каждый 4-битовый регистр содержит число в диапазоне 0-15, которое представляет реальный цвет, выводимый, когда оператор программы встречает один из кодовых номеров. Например, если в каком-то месте программы указано, что символ должен выводиться с кодовым номером 0, то цвет выводимого символа определяется кодом цвета, хранящимся в регистре палитры 0. Начальное значение этого регистра 0000, поэтому будет выводиться черный цвет. Но содержимое этого регистра может быть изменено, скажем на 0001, а в этом случае кодовый номер 0 приведен к выводу синим цветом. Кодовые номера, используемые в регистрах палитры, такие же, как и в операторах программы. На рис. 4.1 показана начальная установка регистров палитры для всех регистров, кроме регистра для зеленого цвета, который изменен так, чтобы выводился цвет магента.

Чтобы запрограммировать регистр палитры PCjr, нужно сначала послать его номер (от 10H до 1FH) в массив ворот дис-

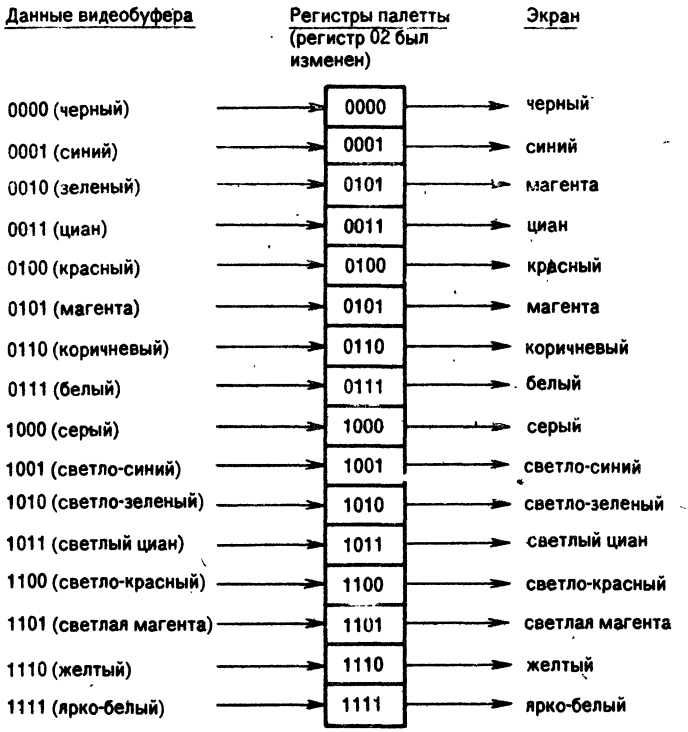


Рис. 4.1. Замена зеленого цветом магента

плея, адрес порта которого 3DAH. Затем нужно послать данные по тому же адресу. Чтобы быть уверенным, что массив готов принять номер регистра, а не данные, надо сначала прочитать из порта 3DAH, отбросив прочитанное.

EGA также использует 16 регистров палитры. Они расположены в порте с номером 3C0H, а номера их меняются от 00 до 0FH. Надо сначала прочитать из порта 3DAH, чтобы переключить порт на его адресный регистр, затем послать номер регистра палитры в 3C0H, а затем послать данные. Когда переключатели на EGA установлены на улучшенный режим (для улучшенного цветного дисплея IBM), палитра может быть выбрана из 64 цветов. В

этом случае установка регистра палитры имеет длину 6 бит в формате R'G'B'RGB. Биты RGB дают темные цвета, а биты R'G'B' - цвета позышенной яркости. Когда установлены и R', и R, например, это приводит к очень яркому красному цвету. Биты могут смешиваться, давая новые оттенки. Если регистры палитры, предназначенные для 64 цветов, используются не в улучшенном режиме, то 4-й и 5-й биты регистра игнорируются и содержимое регистров рассматривается по обычной схеме RGB. Поскольку PCjr и EGA используют регистры палитры, выбор фонового цвета не ограничен битом 7 байта атрибутов в качестве бита мигания.

Монохромные символы

Монохромные символы используют байт атрибутов несколько более необычным образом. Как и с атрибутами цвета, биты 0-2 устанавливают основной цвет, а биты 4-6 - фоновый. Эти цвета могут быть только белым и черным со следующим соответствием битам:

Бит	Бит	Бит	Атрибут	
6 или 2	5 или 1	4 или 0	Основной	Фоновый
0	0	0	черный	черный
0	0	1	подчеркнутый белый	белый
0	1	0	белый	белый
0	1	1	белый	белый
1	0	0	белый	белый
1	0	1	белый	белый
1	1	0	белый	белый
1	1	1	белый	белый

Нормальный режим - белый на черном, когда биты 0-2 установлены в 111, а биты 4-6 установлены в 000. Негативное изображение создается обратными значениями битов. Символы выводятся с повышенной яркостью, когда бит 3 установлен в 1; способа придать повышенную яркость фону, когда символы выводятся в негативном изображении, не существует, недоступно и подчеркивание в негативе. Во всех случаях установка в 1 бита 7 дает мигание символов. Возможны только 10 комбинаций, когда символы видны. Они могут быть реализованы различными установками битов. Ниже приводятся по одной из возможных установок для каждого случая:

Атрибут	Цепочка битов	Шестнадцатиричное	Десятичное
нормальный	00000111	7	7
интенсивный	00001111	F	15
нормальный подчеркнутый	00000001	1	1
интенсивный подчеркнутый	00001001	9	9
негативный	01110000	70	112
нормальный мигающий	10000111	87	135
интенсивный мигающий	10001111	8F	143
нормальный мигающий подчеркнутый	10000001	81	129
яркий мигающий подчеркнутый	10001001	89	137
яркий негативный	11110000	F0	240

Высокий уровень

Бейсик устанавливает цвета и атрибуты символов оператором COLOR. Все операторы PRINT и WRITE, которые следуют за данным оператором COLOR, выполняются с атрибутами, указанными в этом операторе. Цвет фона меняется только для выводимых символов, но не для всего экрана. Новый оператор COLOR не влияет на то, что было выведено ранее.

За исключением монохромного адаптера, COLOR 3,4 устанавливает основной цвет символа циан (#3), а фоновый - красный (#4). Диапазон кодов основных цветов 0 - 31, причем числа 0 - 15 соответствуют цветам, перечисленным в вышеприведенной таблице, а числа 16 - 31 получаются прибавлением к любому из этих кодов числа 16, что дает тот же цвет, но с миганием символов. (При мигании основной цвет периодически меняется на фоновый, в то время как фоновый цвет остается неизменным.)

Операторы PRINT и WRITE могут также выводить символы на графический экран. При этом цвет символов - это всегда третий цвет текущей палитры, т.е. желтый/коричневый для палитры 0 и белый для палитры 1.

Отметим, что когда вы начинаете работать в цветном текстовом режиме, весь экран черно-белый. Чтобы закрасить весь экран в фоновый цвет, необходимо указать оператором COLOR ,2, например, зеленый цвет и затем очистить экран командой CLS. Когда вы чистите экран по ходу выполнения программы, необходимо, чтобы последний оператор COLOR установил фоновый цвет таким, каким вы хотите закрасить весь экран.

Для монохромного дисплея атрибуты устанавливаются аналогичным образом. 0 соответствует черному цвету, а любое из чисел

1 соответствует белому. Таким образом, COLOR 0,7 устанавливает черное изображение на белом фоне (негатив), в то время как COLOR 7,0 дает вывод белых символов на черном фоне (обычная установка). Имеется одно исключение: если в качестве основного цвета использовать код 1, то будут выводиться подчеркнутые символы. Прибавив 8 к любому из кодов основного цвета, получим яркое изображение. Прибавив 16 к любому из кодов 0-15, получим мигающие символы. Таким образом, 7 + 8 + 16 = 31 дает яркое мигающее белое изображение. Для фонового цвета допустимы только значения от 0 до 7.

Если вы используете прямое отображение в память [4.3.1], то оператор COLOR не влияет на вывод. Вместо этого вы должны выбрать требуемую установку атрибутов из таблиц и присвоить значение соответствующего байта атрибутов оператором POKE. Помните, что байты атрибутов всегда занимают нечетные позиции в видеобуфере. Отображение в память позволяет иметь 16 фоновых цветов в Бейсике (при условии, что вам не нужны мигающие символы). Для графического адаптера введите OUT &H3D8,8, чтобы старший бит каждого атрибута действовал как бит яркости для фоновых цветов. В следующем примере в центре экрана печатается ярко-красный "!" на светло-красном фоне.

```
100 DEF SEG = &HB800          'указываем на буфер цветного дисплея
110 OUT &H3D8,8               'используем 16 фоновых цветов
120 POKE 1000,33              'печатаем ! в центре экрана
130 POKE 1001,196             'красный на светло-красном (11000100)
```

Как уже говорилось, PCjr хранит бит мигания в массиве ворот дисплея. Вот та же программа для PCjr (но она не будет работать в режиме двухцветной графики):

```
100 DEF SEG = &HB800          'указываем на видеобуфер
110 X = INP(&H3AH)            'читаем из массива ворот дисплея
120 OUT &H3AH,3               'требуем доступ к регистру 3
130 OUT &H3AH,0               'сбрасываем все биты этого регистра
140 POKE 1000,33              'печатаем ! в центре экрана
150 POKE 1001,196             'красный на светло-красном (11000100)
```

Приведем еще пример изменения назначения цвета регистра палитры. Код цвета, который обычно выводит синий (0001), пусть выводит цвет магента (0101). Номер регистра массива ворот дисплея, соответствующий коду цвета 1, - 11H.

```
100 X = INP (&H3AH)           'читаем из массива ворот дисплея
110 OUT &H3AH,&H11            'требуем доступ к регистру 11H
120 OUT &H3AH,5               'помещаем туда код магенты (0101 = 5).
```

Средний уровень

Прерывания DOS и BIOS предоставляют очень скудные возможности для работы с цветным текстом. Только функция 9 прерывания 10H принимает байт атрибутов при выводе символа. Функция A прерывания 10H выводит символ без указания цвета или атрибута; она просто помещает символ в видеобуфер, не трогая байт атрибута, таким образом, атрибуты сохраняют свое старое значение. Функция D прерывания 10H также оставляет нетронутым байт атрибутов. Все эти функции обсуждаются в [4.3.1].

Функции вывода на экран DOS прерывания 21H всегда выводят белое на черном. Даже если для всего экрана установлен некоторый фоновый цвет, функции DOS устанавливают атрибут в нормальный черный при выводе каждого символа. Однако имеется способ преодолеть это ограничение. MS-DOS предоставляет драйвер устройства ANSI.SYS, который может интерпретировать специальные Esc-последовательности. В приложении Д объясняются основы его использования. Esc-последовательности выводятся через функцию 9 прерывания 21H, которая обычно печатает строку символов на экране. В этом случае строка состоит из символа Esc, за которым следует [, а далее одно или более кодовых чисел из приведенного ниже списка. Строка должна кончаться символом m и обычным ограничителем $. Вот кодовые номера:

0	все атрибуты выключены (черный на белом)		
1	включена повышенная интенсивность		
4	включено подчеркивание		
5	включено мигание		
7	включено негативное изображение		
8	все включено (при этом символы невидимы)		
30	черный основной цвет	40	черный фон
31	красный основной цвет	41	красный фон
32	зеленый основной цвет	42	зеленый фон
33	желтый основной цвет	43	желтый фон
34	синий основной цвет	44	синий фон
35	основной цвет магента	45	фон магента
36	основной цвет циан	46	фон циан
37	белый основной цвет	47	белый фон

Отметим, что когда функции MS-DOS выводят символы в графическом режиме, они обычно используют код 3 текущей палитры. С помощью Esc-последовательностей можно установить

цвет символа соответствующим любому из цветов палитры. Надо указывать 30 или 31 для фонового цвета, 32 или 33 - для кода 1, 34 или 35 - для кода 2 и 36 или 37 - для кода 3. В этом случае не надо указывать фоновый цвет.

В следующем примере на экран выводятся две строки с помощью функции 9 прерывания 21H. Первая выводится синим на красном, а вторая - мигающим цианом на красном. Не надо переопределять красный в качестве фонового цвета для второй строки, поскольку назначения цветов действуют на все последующие команды вывода (включая функции BIOS прерывания 10H), до тех пор, пока не будут сделаны другие назначения. Отметим, как просто перемешивать команды управления цветом с выводом самих строк.

```
;---в сегменте данных
STRING_1 DB   'The rain in Spain',0AH,0DH,'$'
STRING_2 DB   'Falls mainly on the plain$'
BLUE_REDDB    27,'[34;41m$'
BLI_CYAN DB   27,'[5;36m$'
;---вывод строк
         MOV  AH,9            ;функция вывода строки
         LEA  DX,BLUE_RED     ;адрес управляющей строки в DX
         INT  21H             ;все будет выдаваться синим на красном
         LEA  DX,STRING_1     ;указываем на первую строку
         INT  21H             ;печатаем строку
         LEA  DX,BLI_CYAN     ;адрес второй управляющей строки
         INT  21H             ;меняем цвет на мигающий циан
         LEA  DX,STRING_2     ;указываем на вторую строку
         INT  21H             ;печатаем строку
```

Вы всегда должны заботиться о том, чтобы сбросить атрибуты цвета в нормальное состояние перед завершением программы, поскольку в противном случае они будут действовать и на вывод последующих программ. В конце следует вывести Esc-последовательность, использующую код номер 0, как указано выше.

PCjr и EGA имеют специальную функцию BIOS для установки содержимого регистров палитры. Это подфункция 0 функции 10H прерывания 10H. Надо поместить номер регистра палитры (от 0 до 15) в BL, а значение кода цвета (также от 0 до 15) - в BH, а затем выполнить прерывание. Подфункция 2 функции 10H устанавливает все регистры палитры, а также цвет границы, используя 17-байтовый массив, на который должны указывать ES:DX. Байты 0-15 массива помещаются в регистры палитры 0-15,

а байт 16 устанавливает цвет границы. О том, как отдельно установить цвет границы, см. [4.1.4].

Низкий уровень

Как уже объяснялось в описании примера на Бейсике, надо поместить требуемое значение байта атрибутов в видеобуфер за тем символом, к которому эти атрибуты должны относиться. Приведем пример для цветного адаптера или PCjr. В примере устанавливается текстовый экран 80*25 с 16 фоновыми цветами, а затем экран инициализируется в красный цвет на светло-синем фоне:

```
;---установка 16 фоновых цветов в текстовом режиме 80*25
        MOV  AL,00001001B  ;установка в 0 бита мигания
        MOV  DX,3D8H       ;адрес регистра
        OUT  DX,AL         ;посылаем в регистр
;---инициализируем весь экран в красный на светло-синем фоне
        MOV  AX,0B800H     ;указываем на видеобуфер
        MOV  ES,AX
        MOV  CX,2000       ;записываем атрибут в 2000 ячеек
        MOV  BX,1          ;BX указывает на байт атрибутов
        MOV  AL,10010100B  ;значение байта атрибутов
NEXT:   MOV  ES:[BX],AL    ;посылаем атрибуты в буфер
        INC  BX            ;увеличиваем указатель на атрибуты
        INC  BX
        LOOP NEXT          ;пишем в следующую позицию
```

4.1.4. Установка цвета границы экрана

Граница символьного экрана может иметь цвет, отличный от фонового цвета центральной части экрана. Может быть использован любой из 16 цветов. С другой стороны, графические экраны технически не имеют области границы. Когда цвет фона устанавливается в графическом режиме, весь экран, включая область границы, окрашивается в этот цвет. Однако операции вывода точек на экран не имеют доступа к области границы; если большую часть адресуемых точек экрана изменить в нефоновый цвет, то будет создана видимость границы экрана.

Высокий уровень

Третий параметр оператора Бейсика COLOR устанавливает цвет границы. Используются те же кодовые номера цветов, приведенные в [4.1.3]. Например, для установки границы в светло-

синий цвет надо написать COLOR ,,8. PCjr, кроме того, может изменять цвет за счет изменения установки регистра палитры, соответствующего коду цвета, указанного для цвета границы. Полное объяснение см. в [4.1.3].

Средний уровень

Для всех видеосистем фоновый цвет может быть установлен функцией BH прерывания 10H. Эта функция устанавливает также основные цвета. Чтобы указать, что надо изменить фоновый цвет, помещают 0 в BH, а код цвета - в BL и выполняют прерывание. Кроме того, PCjr и EGA имеют специальную функцию для установки фонового цвета. Это подфункция 1 функции 10H прерывания 10H. Надо поместить 10H в AH, 1 в AL и код цвета в BH. Никаких значений не возвращается.

Низкий уровень

Для цветного графического адаптера биты 0-3 порта 3D9H (регистр выбора цвета) устанавливают цвет границы, когда экран находится в текстовом режиме. Как обычно, назначение битов в восходящем порядке - синий (B), зеленый (G), красный (R) и интенсивность. Поскольку этот адрес предназначен только для записи, все остальные биты этого регистра должны быть правильно установлены. Это бит 4, который, если его установить в 1, приводит к тому, что все фоновые цвета будут выводиться с высокой интенсивностью.

```
;---установка светло-синего цвета границы
MOV  AL,00001001B      ;атрибут светло-синего цвета
MOV  DX,3D9H           ;адрес регистра выбора цвета
OUT  DX,AL             ;устанавливаем цвет границы
```

Для PCjr массив ворот дисплея [4.1.1] имеет регистр, который устанавливает цвет границы. Это 4-битовый регистр, причем биты 0-3 соответствуют синему, зеленому, красному и высокой интенсивности, когда установлены в 1. Для установки светло-синего цвета надо послать в регистр 1001. Регистр цвета границы - это регистр 2 массива ворот дисплея. Чтобы получить доступ к этому регистру, надо сначала послать 2 в порт по адресу 3DAH. Затем надо послать данные по тому же адресу. Чтобы быть уверенным, что микросхема готова принять номер регистра, а не данные, надо

сначала прочитать из порта 3DAH. Следующий пример устанавливает красный цвет границы (бит 2 установлен):

```
MOV  DX,3DAH      ;адрес порта массива ворот дисплея
IN   AL,DX        ;чтение для подготовки микросхемы
MOV  AL,2         ;номер требуемого регистра
OUT  DX,AL        ;посылаем в порт
MOV  AL,4         ;устанавливаем только бит 2
OUT  DX,AL        ;устанавливаем цвет границы
```

Для EGA цвет границы устанавливается регистром сканирования (overscan). Это регистр номер 11H порта с адресом 3C0H. Надо сначала прочитать из этого порта, чтобы переключить его на адресный регистр, затем послать туда номер 11H в качестве индекса, а затем послать данные. Имеют значение только младшие 4 бита данных, если только EGA не связан с улучшенным цветным дисплеем IBM, а в этом случае имеют значение младшие 6 бит, которые устанавливают цвет границы.

4.1.5. Очистка части/всего экрана

Очистка экрана состоит в записи пробела в каждую из позиций экрана (код ASCII - 32). Однако если при выводе на экран были использованы ненормальные атрибуты, то должны быть также изменены и байты атрибутов. Операционная система обеспечивает простой способ очистки только части экрана.

Высокий уровень

В Бейсике для очистки экрана применяется оператор CLS. При этом 25-я строка внизу экрана становится пустой, только если был убран список значений функциональных клавиш с помощью команды KEY OFF. Байты атрибутов устанавливаются равными ASCII 7. В [4.5.1] дана процедура прокрутки, которая в Бейсике служит для очистки окон на экране.

Средний уровень

Операционная система предоставляет несколько способов очистки экрана. Ваш выбор зависит от того, какие средства требуются программе для достижения других целей. Первый метод это просто сброс режима дисплея с помощью функции 0 прерывания 10H [4.1.2]. Для символьного экрана каждая позиция заполняется

пробелом (ASCII 32), а все атрибуты устанавливаются нормальными (ASCII 7). Обычно этот метод хорош только в начале программы, когда все равно надо устанавливать режим работы дисплея. Для цветного графического адаптера и PCjr реинициализация режима дисплея приводит к катавасии на экране. Этот эффект отсутствует у монохромного адаптера и EGA.

```
;---очистка экрана путем установки нового режима
MOV  AH,0          ;номер функции установки режима дисплея
MOV  AL,2          ;код режима 80*25 черно-белого
INT  10H           ;очистка экрана
```

Второй метод состоит в использовании функций 6 и 7 прерывания 10H, которые сдвигают экран. Число строк, на которое надо сдвинуть экран, помещается в AL, и когда это число равно нулю, экран очищается. Прерывание позволяет сдвигать только часть экрана, поэтому таким образом можно очистить отдельное окно на экране. Надо поместить координаты левого верхнего угла окна в CX, а координаты правого нижнего угла в DX (номер строки в CH/DH, а номер столбца в CL/DL). Поместите атрибут, с которым должен чиститься экран, в BH. Координаты отсчитываются от 0.

```
;---очистка окна между 3,4 и 13,15
MOV  AH,6          ;используем процедуру сдвига
MOV  AL,0          ;число строк сдвига делаем равным нулю
MOV  BH,7          ;байт атрибутов для заполнения
MOV  CH,3          ;строка для верхнего левого угла
MOV  CL,4          ;столбец для левого верхнего угла
MOV  DH,13         ;строка для нижнего левого угла
MOV  DL,15         ;столбец для нижнего левого угла
INT  10H           ;чистим окно
```

Третий метод заключается в использовании функции 9 прерывания 10H, которая выводит символ и атрибуты столько раз, сколько указано в CX. Значение 2000 чистит весь экран, если курсор был установлен в 0,0 методом, показанным в [4.2.1]. AH должен содержать символ пробела, AL - байт атрибутов, а BH - номер страницы дисплея.

```
;---установка курсора в левый верхний угол экрана
MOV  AH,2          ;функция установки курсора
```

```
MOV  BH,0              ;номер страницы
MOV  DX,0              ;координаты 0,0
INT  10H               ;устанавливаем курсор
;---вывод символа пробела 2000 раз
MOV  AH,9              ;номер функции
MOV  CX,2000           ;число повторений вывода
MOV  AL,' '            ;символ пробела в AL
MOV  BL,7              ;атрибуты в BL
INT  10H               ;очистка экрана
```

Наконец, DOS обеспечивает очистку экрана с помощью специальных Esc-последовательностей, которые работают с драйвером ANSI.SYS. Основные сведения о нем приведены в приложении Д. Эти последовательности - строки, начинающиеся с символа Esc, а завершающиеся ограничителем $. Такие строки выводятся функцией 9 прерывания 21H, при этом DS:DX должны указывать на первый символ строки. DOS интерпретирует строку, не выводя ее на дисплей. Чтобы стереть весь экран, строка должна быть [2J. Чтобы стереть конец строки, начиная от позиции курсора (включая эту позицию), строка должна быть [K.

```
;---в сегменте данных
CLE_LIN DB   27,'[K$'
;---очистка конца строки от позиции курсора
        MOV  AH,9          ;функция вывода строки
        LEA  DX,CLE_LIN    ;DX должен указывать на начало строки
        INT  21H           ;стираем конец строки
```

Низкий уровень

На низком уровне надо просто поместить символы пробела и требуемый байт атрибутов в память дисплея, используя инструкцию STOSW. Вот пример для монохромного дисплея:

```
        MOV AX,0B000H  ;указываем на память дисплея
        MOV ES,AX
        MOV DI,0       ;DI указывает на начало буфера
        MOV AL,32      ;символ пробела
        MOV AH,7       ;нормальные атрибуты
        MOV CX,2000    ;число повторений
REP     STOSW          ;посылаем AX в ES:DI 2000 раз
```

4.1.6. Переключение между видеоадаптерами

Машина может быть оснащена и монохромным, и цветным адаптером или одним из этих адаптеров и EGA. Программа может выбирать, какой из мониторов должен быть активным, изменяя значения битов 4 и 5 в ячейке памяти 0000:0410. Установив оба бита в 1, мы выбираем монохромный адаптер. Изменив установку битов 5-4 на 10, устанавливаем графический адаптер в режиме 80 символов в строке, а на 01 - 40 символов в строке. И наконец, изменив биты на 00, выбираем EGA. Во всех случаях вы должны немедленно подать команду установки режима, поскольку BIOS имеет еще очень много регистров, которые надо изменить, прежде чем дисплей будет работать нормально.

Отметим, что хотя операционная система не может управлять одновременно двумя мониторами, программы могут осуществлять вывод на оба дисплея, используя прямое отображение в память [4.3.1] для адресов буфера неактивного монитора.

Высокий уровень

В Бейсике надо применить следующий код:

```
100 'Переключение на монохромный дисплей
110 KEY OFF: CLS
120 WIDTH 40
130 DEF SEG = 0
140 M = PEEK(&H410)
150 POKE &H410,M OR &H30
160 WIDTH 80
170 LOCATE,,1,12,13
180 KEY ON
```

```
100 'Переключение на цветной графический дисплей (80 символов)
110 KEY OFF: CLS
120 WIDTH 80
130 DEF SEG = 0
140 M = PEEK(&H410)
150 POKE &H410,(M AND &HCF) OR &H20
160 WIDTH 80
170 SCREEN 0
180 LOCATE,,1,6,7
```

```
190 KEY ON

100 'Переключение на EGA (80 символов)
110 KEY OFF: CLS
120 WIDTH 80
130 DEF SEG = 0
140 M = PEEK(&H410)
150 POKE &H410,M AND &HCF
160 WIDTH 80
170 SCREEN 0
180 LOCATE,,1,6,7
190 KEY ON
```

Измените команды WIDTH и SCREEN, чтобы переключиться на другие начальные режимы дисплея.

Низкий уровень

В ассемблере, как и в Бейсике, надо непосредственно изменить биты 4 и 5 по адресу 0000:0410 и сбросить режим дисплея сразу вслед за изменением.

```
;---переключение на монохромный монитор
SUB   AX,AX             ;обнуляем AX
MOV   ES,AX             ;устанавливаем ES на начало памяти
MOV   DL,ES:[410H]      ;получаем байт по адресу 0000:0410
OR    DL,00110000B      ;устанавливаем биты 4 и 5
MOV   ES:[410H],DL      ;возвращаем байт
MOV   AH,0              ;функция установки режима дисплея
MOV   AL,0              ;монохромный режим 80*25
INT   10H               ;устанавливаем режим

;---переключение на цветной монитор (40 символов)
SUB   AX,AX             ;устанавливаем ES на начало памяти
MOV   ES,AX             ;
MOV   DL,ES:[410H]      ;берем байт по адресу 0000:0410
AND   DL,11001111B      ;сбрасываем биты 4 и 5
OR    DL,00010000B      ;устанавливаем бит 4
MOV   ES:[410H],DL      ;возвращаем байт
MOV   AH,0              ;функция установки режима дисплея
```

```
MOV  AL,1              ;цветной режим 40*25
INT  10H               ;устанавливаем режим

;---переключение на EGA
SUB  AX,AX             ;устанавливаем ES на начало памяти
MOV  ES,AX             ;
MOV  DL,ES:[410H]      ;берем байт по адресу 0000:0410
AND  DL,11001111B      ;сбрасываем биты 4 и 5
MOV  ES:[410H],DL      ;возвращаем байт
MOV  AH,0              ;функция установки режима дисплея
MOV  AL,1              ;цветной режим 40*25
INT  10H               ;устанавливаем режим
```

4.2. Управление курсором

Курсор служит двум целям. Во-первых, он служит указателем места на экране, в которое операторы программы посылают свой вывод. Во-вторых, он обеспечивает видимую точку отсчета на экране для пользователя программы. Только для второго применения курсор должен быть видимым. Когда курсор невидим (выключен), он все равно указывает на позицию экрана. Это важно, поскольку любой вывод на экран, поддерживаемый операционной системой, начинается с текущей позиции курсора.

Курсор генерируется микросхемой контроллера дисплея 6845, описанной в [4.1.1]. Эта микросхема имеет регистры, устанавливающие размер и положение курсора. Микросхема 6845 обеспечивает только мерцающий курсор, хотя имеются программные способы создания немерцающего курсора [4.2.6]. Частота мерцания курсора не может быть изменена. В графических режимах курсор не выводится, хотя символы позиционируются на экране теми же процедурами установки курсора, что и в текстовых режимах.

Когда видеосистема работает в режиме, допускающем несколько дисплейных страниц, каждая страница имеет свой собственный курсор и при переключении между страницами восстанавливается позиция курсора, которую он занимал, когда было последнее обращение к восстанавливаемой странице. Некоторые режимы дисплея позволяют иметь до 8 дисплейных страниц и соответствующие им позиции курсора хранятся в наборе восьми 2-байтовых переменных в области данных BIOS, начиная с адреса 0040:0050H. В каждой переменной младший байт содержит номер

столбца, отсчитывая от 0, а старший байт содержит номер строки, также отсчитывая от 0. Когда число страниц меньше 8, используются переменные, расположенные в более младших адресах памяти.

4.2.1. Установка курсора в абсолютную позицию

Для курсора могут быть установлены абсолютные координаты или координаты относительно его текущей позиции [4.2.2]. Абсолютные координаты могут меняться в пределах 25 строк и 80 (иногда 40) столбцов. В языках высокого уровня обычно отсчитывают координаты экрана начиная с 1, и, таким образом, позиция левого верхнего угла - 1,1. В ассемблере отсчет всегда начинается с нуля и позиция левого верхнего угла - 0,0.

Высокий уровень

Бейсик нумерует строки от 1 до 25, а столбцы - от 1 до 80. Формат оператора LOCATE, который устанавливает позицию курсора, такой: LOCATE строка,столбец. Если курсор не устанавливается, то он переходит в первую позицию строки после ввода возврата каретки, а сдвиг экрана начинается после того, как будет заполнена 24-я строка. Чтобы вывести в 25-ю строку, вы должны использовать LOCATE (предварительно очистив эту строку с помощью KEY OFF). Для отмены автоматического сдвига экрана в строках 24 и 25 надо завершать оператор PRINT точкой с запятой (чтобы отменить сдвиг в позициях 24,80 и 25,80, надо использовать прямое отображение в память [4.3.1]). Ниже приведен пример рисования вертикальной черты с помощью одного из символов псевдографики в центре экрана:

```
100 FOR N = 1 TO 25          'повтор для каждой строки
110 LOCATE N,40              'установка курсора в середину строки
120 PRINT CHR$(186);         'печатаем вертикальную черту
130 NEXT                     'переход к следующей строке
```

Когда используется несколько дисплейных страниц, оператор LOCATE действует на текущей активной странице памяти. Если страница, выводимая в данный момент на монитор, не активна, то положение курсора на экране не меняется. Отметим, что Бейсик имеет собственную переменную, хранящую текущее положение курсора. Если вы подключите ассемблерную подпрограмму, кото-

рая изменит положение курсора, то Бейсик проигнорирует новую позицию курсора, когда ему будет возвращено управление.

Средний уровень

Операционная система предоставляет два способа позиционирования курсора в абсолютную позицию на экране. Функция 2 прерывания 10H устанавливает курсор, относящийся к указанной странице памяти. Страницы нумеруются начиная с нуля и для монохромного дисплея номер страницы (находящийся в BH) должен всегда быть равным 0. DH:DL содержат строку и столбец, которые тоже нумеруются с 0. Курсор меняет свое положение на экране, только если установка курсора относится к текущей активной странице.

```
;---установка курсора в строку 13, столбец 39
MOV  AH,2          ;номер функции
MOV  BH,0          ;номер страницы
MOV  DH,13         ;строка
MOV  DL,39         ;столбец
INT  10H           ;позиционируем курсор
```

Второй метод позиционирования курсора состоит в применении специального драйвера устройства ANSI.SYS, который должен быть загружен при старте системы. В приложении Д даны необходимые сведения. Для посылки команды, содержащей информацию о строке и столбце, служит функция 9 прерывания 21H. Строка начинается с символа Esc (ASCII 27), а завершается символом-ограничителем $. Формат строки - Esc [строка, столбецH$, где строка и столбец нумеруются от нуля, а Esc обозначает код ASCII 27. Например, строка 27,'10;60H$' устанавливает курсор в строку 10, столбец 60.

Хотя такой метод кажется излишне сложным, он оказывается очень удобным при выводе ряда строк на экран, так как Esc-последовательность обрабатывается как одна из строк набора. В данном примере три строки сообщения разбросаны по всему экрану.

```
;---в сегменте данных
POS_1 DB   27,'[10;30H$'
STR_1 DB   'There are two options:$'
POS_2 DB   27,'[13;32H$'
STR_2 DB   '(1) Review part 1$'
POS_3 DB   27,'[15;32H$'
```

```
STR_3 DB   '(2) Move on to part 2$'
;---печать строк
      MOV  AH,9       ;номер функции вывода строки
      LEA  DX,POS_1   ;1-я строка позиционирования курсора
      INT  21H        ;позиционируем курсор
      LEA  DX,STR_1   ;1-я текстовая строка
      INT  21H        ;вывод строки
      LEA  DX,POS_2   ;и т.д.
      INT  21H
      LEA  DX,STR_2
      INT  21H
      LEA  DX,POS_3
      INT  21H
      LEA  DX,STR_3
      INT  21H
```

Низкий уровень

Регистры 14 и 15 микросхемы 6845 хранят положение курсора. Вы можете изменить их значение и курсор передвинется в соответствующую позицию экрана, но прерывания вывода на экран DOS и BIOS будут игнорировать вашу установку и вернут курсор в старое положение. Это происходит потому, что каждый раз при вызове этих прерываний они восстанавливают регистры курсора, используя 2-байтовое значение, хранящееся в области данных BIOS. В этой области, начиная с адреса 0040:0050, могут находиться до восьми таких значений, дающих текущее положение курсора для каждой из страниц дисплея. Процедура низкого уровня должна модифицировать и эти значения, чтобы изменить состояние курсора полностью.

Позиция курсора хранится в регистрах 14 и 15 как число от 0 до 1999, что соответствует 2000 (25x80) позициям экрана. Не спутайте эту систему нумерации с позициями видеобуфера от 0 до 3999, где каждый символ сопровождается еще байтом атрибутов (для получения эквивалентного указателя на позицию курсора надо сдвинуть указатель видеобуфера на 1 бит вправо). Обращаем также ваше внимание на то, что не надо менять местами старший и младший байты: в регистре 14 - старший, а 15 - младший.

```
;---в программе
      MOV  BL,24      ;строка в BL (0-24)
      MOV  BH,79      ;столбец в BH (0-79)
      CALL SET_CUR    ;вызов процедуры
;---процедура установки курсора
```

```
SET_CUR PROC
;получаем доступ к регистру младшего байта
        MOV  DX,3B4H  ;порт адресного регистра 6845
        MOV  AL,15    ;выбираем регистр 15
        OUT  DX,AL    ;посылаем запрос
;вычисление позиции курсора
        MOV  AL,80    ;умножаем номер строки на 80
        MUL  BL       ;в AX - номер строки, умноженный на 80
        MOV  BL,BH    ;переносим номер столбца в BL
        SUB  BH,BH    ;распространяем BL на BX
        ADD  AX,BX    ;вычисляем позицию курсора
;посылаем младший байт результата
        INC  DX       ;адресуем управляющий регистр
        OUT  DX,AL    ;посылаем младший байт
;получаем доступ к регистру старшего байта
        MOV  AL,14    ;номер требуемого регистра
        DEC  DX       ;восстанавливаем порт адресного регистра
        OUT  DX,AL    ;посылаем запрос
;посылаем старший байт результата
        INC  DX       ;адресуем управляющий регистр
        MOV  AL,AH    ;помещаем старший байт в AL
        OUT  DX,AL    ;посылаем старший байт
        RET
SET_CUR ENDP
```

4.2.2. Относительное позиционирование курсора

Иногда полезно сдвинуть курсор относительно его предыдущей позиции: на строку вверх, на три столбца вправо и т.д. Достаточно просто использовать для этой цели уже описанное абсолютное позиционирование курсора. Но для удобства MS-DOS предоставляет некоторые возможности относительного перемещения курсора.

Средний уровень

Функции относительного перемещения курсора выполняются Esc-последовательностями. Это строки, которые выводятся на экран с помощью функции 9 прерывания 21H. В приложении Д даны основы их использования. Такие последовательности интерпретируются MS-DOS как команды перемещения курсора, а не вы-

вод символов строки. Строка начинается с символа Esc (ASCII 27), затем идет символ [, а символ $ отмечает конец строки. Сама строка состоит из числа позиций, на которое надо сдвинуться, и кода направления. Сдвиг на 3 позиции:

вверх	3A
вниз	3B
вправо	3C
влево	3D

Числа записываются как коды ASCII. Не преобразуйте, например, 33C (33 пробела вправо) в 33,'C'; должно быть '33C'. В приведенном примере цифры 1-8 помещаются через определенные интервалы поперек экрана, как метки столбцов данных. Промежутки между цифрами генерируются Esc-последовательностями, которые сдвигают курсор вправо после вывода каждой цифры.

```
;---в сегменте данных
CUR_RITDB    27,'[9C$'
;---установка начальной позиции курсора
        MOV  BH,0          ;номер страницы
        MOV  DH,1          ;строка
        MOV  DL,5          ;столбец
        MOV  AH,2          ;функция установки курсора
        INT  10H           ;установка курсора
;---вывод цифр
        LEA  BX,CUR_RIT ;BX будет обмениваться с DX
        MOV  CX,8          ;число цифр для вывода
        MOV  DL,'0'        ;начинаем с 0
NEXT:   MOV  AH,2          ;функция DOS для вывода символа
        INT  21H           ;выводим символ
        INC  DL            ;переходим к следующему коду ASCII
        XCHGDX,BX          ;помещаем указатель на строку в DX
        MOV  AH,9          ;функция вывода строки
        INT  21H           ;сдвигаем курсор на 9 позиций вправо
        XCHGDX,BX          ;возвращаем в DX код ASCII
        LOOP NEXT          ;переходим к следующей цифре
```

Существует также пара Esc-последовательностей, которые управляют переносом курсора на следующую строку при достижении им конца текущей строки. Когда устанавливается отсутствие переноса, лишние символы при выводе отбрасываются. Строка, запрещающая перенос, Esc [= 7h (27,'[= 7h'). Для возврата к режиму автоматического переноса на следующую строку используется строка Esc [= 7l (27,'[= 7l').

4.2.3. Включение и выключение курсора

Курсор генерируется микросхемой 6845. Он функционирует совершенно независимо от видеопамяти. Это значит, что при прямой адресации в память дисплея [4.3.1] программное обеспечение должно координировать перемещения курсора с вставкой нового символа в буфер. Отметим, что микросхема 6845 не может ни создавать немерцающий курсор, ни изменять частоту его мерцания. В [4.2.6] показано, как сконструировать другие "искусственные" типы курсора.

Высокий уровень

Интерпретатор Бейсика автоматически выключает курсор при запуске программы. Курсор появляется, когда используется оператор INPUT. Если вашей программе необходим курсор, скажем для процедуры INKEY$, то он должен быть включен установкой третьего параметра оператора LOCATE в 1 (0 снова выключит его). Напоминаем, что первые два параметра оператора LOCATE устанавливают строку и столбец, в которых должен выводиться курсор.

```
100 LOCATE 15,40,1      ;включить курсор, его позиция 15,40 или
100 LOCATE ,,1          ;включить курсор в текущей позиции и
100 LOCATE ,,0          ;снова выключить курсор
```

Курсор будет оставаться при последующих появлениях оператора LOCATE без установки каждый раз третьего параметра. Однако надо отметить, что операторы INPUT и INPUT$ выключат его после их выполнения.

Средний уровень

Ассемблерные программы оставляют курсор включенным до тех пор, пока им не указано обратное. Операционная система не предоставляет специальных средств выключения курсора, но это легко сделать. Надо просто позиционировать курсор за пределы экрана, с помощью функции 2 прерывания 10H установить его в первую позицию 26-й строки. Помните, что координаты отсчитываются от нуля, так что этой позиции соответствуют координаты 25,0.

```
MOV  BH,0               ;номер страницы (всегда 0 для монохромного)
MOV  DH,25              ;строка
```

```
MOV  DL,0        ;столбец
MOV  AH,2        ;номер функции
INT  10H         ;устанавливаем курсор за пределы экрана
```

Низкий уровень

Бит 6 регистра 10 микросхемы 6845 [4.1.1] выключает курсор, когда он установлен в 1, и включает его, когда сброшен в 0. Этот регистр содержит также значение "начальной строки" для курсора, которое вместе со значением "конечной строки" определяет толщину курсора [4.2.4]. Поскольку тип курсора не имеет значения, когда курсор выключен, надо просто поместить в регистр 10 значение 32, чтобы установить бит 6. Для восстановления курсора вы должны также вернуть значение "начальной строки" курсора. В обычном случае это значение равно 11. Значение "конечной строки" при этих процедурах не меняется, поскольку оно хранится в другом регистре.

```
;---выключение курсора
MOV  DX,3B4H     ;номер порта адресного регистра 6845
MOV  AL,10       ;выбор регистра 10
OUT  DX,AL       ;посылаем запрос
INC  DX          ;доступ к регистру через следующий порт
MOV  AL,32       ;устанавливаем бит 6 для выключения курсора
OUT  DX,AL       ;выключаем курсор
;---обратное включение курсора
MOV  AL,11       ;значение "начальной строки"
OUT  DX,AL       ;включаем курсор
```

4.2.4. Изменение формы курсора

Курсор может меняться по толщине от тонкой линии до максимального размера, отводимого под символ. Он строится из коротких горизонтальных отрезков, верхний из которых называется *начальной строкой* курсора, а нижний - *конечной строкой*. Для монохромного дисплея под каждый символ отводится 14 строк, пронумерованных от 0 до 13 начиная сверху. Промежутки между символами обеспечиваются двумя верхними строками и тремя нижними. Большинство символов располагаются в строках 2-10, хотя хвостики некоторых символов достигают линий 12 и 13, в то время как подчеркивание занимает одну двенадцатую строку.

На 200-строчном цветном дисплее для каждого символа отведится только 8 строк, а символ рисуется в верхних семи строках.

Эти 8 строк пронумерованы от 0 до 7 начиная сверху, и нормальный курсор формируется одной строкой 7. (Отметим, что на цветном дисплее нет подчеркивания, поскольку использование для этого строки 7 привело бы к тому, что символы сливались бы с расположенными под ними.) Цветной дисплей высокого разрешения использует 14-строчный монохромный вариант, когда он работает в режиме высокого разрешения, и 8-строчный вариант при работе в одном из цветных графических режимов.

Курсор может быть сформирован любой комбинацией прилегающих отрезков. Для монохромного дисплея он занимает все отведенное под символ место, когда начальная строка равна 0, а конечная строка равна 13 (для графического дисплея значение конечной строки должно быть равно 7). Если значения начальной и конечной строк совпадают, то возникает однострочный курсор. Если номер конечной строки меньше, чем номер начальной, то возникает курсор, состоящий из двух частей, так как происходит перенос в верхние строки. Например, если начальная строка равна 12, а конечная - 1, то сначала заполняется строка 12, затем - 13, затем - 0 и, наконец, - 1. Курсор при этом принимает форму двух параллельных линий, указывающих верхнюю и нижнюю границы ряда, который он занимает.

BIOS хранит 2-байтовую переменную по адресу 0040:0060, которая содержит текущие значения начальной и конечной строк. Первый байт содержит значение конечной строки, а второй - начальной.

Высокий уровень

В Бейсике оператор LOCATE может не только позиционировать курсор и включать или выключать его, но и управлять его формой. Параметры, устанавливающие начальную и конечную строки, - это 4-е и 5-е числа, следующие за словом LOCATE. Другие параметры могут быть опущены, если присутствуют разделяющие их запятые. Таким образом, чтобы создать толстый курсор, занимающий строки со 2 по 12, надо записать LOCATE ,,,2,12. Отметим, что Бейсик обычно выключает курсор, когда начинает выполнение программы. Как включить его см. в [4.2.3].

Средний уровень

Функция 1 прерывания BIOS 10H устанавливает начальную и конечную строки курсора. В CH должна быть указана начальная, а в CL - конечная строки.

```
;---установка начальной и конечной строк курсора
MOV  AH,1            ;номер функции
MOV  CH,0            ;начать курсор в верхней строке
MOV  CL,7            ;окончить курсор в восьмой строке
INT  10H
```

Низкий уровень

Регистры 10 и 11 контроллера дисплея 6845 содержат значения начальной и конечной строк соответственно. Доступ к обоим регистрам осуществляется через порт 3B5H для монохромного адаптера и через порт 3D5H - для цветного адаптера и PCjr. Предварительно надо послать номер требуемого регистра в адресный регистр, имеющий адрес порта 3B4H (см. [4.1.1]). Значения занимают младший конец каждого регистра. Однако регистр начальной строки (#10) битами 5 и 6 указывает также, должен ли выводиться курсор. Курсор выводится, когда оба эти бита сброшены в 0. Поместив в регистр только номер начальной строки, мы установим эти биты в 0. Остальные биты этого регистра не используются.

```
;---установка начальной строки
MOV  DX,3B4H         ;доступ к адресному регистру 6845
MOV  AL,10           ;выбор регистра 6845
OUT  DX,AL           ;посылка запроса
MOV  AL,0            ;номер начальной строки 0
INC  DX              ;переходим к управляющему регистру
OUT  DX,AL           ;посылаем номер начальной строки
;---установка "конечной строки"
MOV  AL,11           ;выбираем регистр 11
DEC  DX              ;возвращаемся к адресному регистру
OUT  DX,AL           ;посылаем запрос
MOV  AL,7            ;номер конечной строки 7
INC  DX              ;переходим к управляющему регистру
OUT  DX,AL           ;посылаем номер конечной строки
```

4.2.5. Чтение/сохранение/восстановление позиции курсора

Программы иногда читают и сохраняют текущее положение курсора, с тем чтобы можно было временно перевести курсор в командную строку, а затем вернуть его в исходную позицию. Текущая позиция курсора для каждой из доступных страниц хранится в области данных BIOS. Имеется восемь 2-байтовых переменных, размещающихся начиная с адреса 0040:0050. Первая

позиция соответствует странице 0, вторая - странице 1 и т.д. Младший байт каждой переменной содержит номер столбца, а старший - номер строки. Как столбцы, так и строки нумеруются начиная с нуля.

Высокий уровень

В Бейсике оператор CRSLIN возвращает строку, а POS - столбец. Оператор POS должен быть снабжен фиктивным аргументом, т.е. он всегда должен записываться в виде POS(0). В данном примере курсор переводится в нижнюю строку экрана, а затем возвращается на место. Отметим, что курсор возвращается на место после выполнения оператора INPUT [4.2.3].

```
100 ROW = CRSLIN                    'получаем строку курсора
110 COL = POS(0)                    'получаем столбец курсора
120 LOCATE 25,1                     'переводим курсор в командную строку
130 INPUT "Enter file name", F$     'запрос на ввод
140 LOCATE ROW,COL,1                'восстанавливаем позицию курсора
```

Средний уровень

Функция 3 прерывания 10H возвращает строку курсора в DH, а столбец - в DL. На входе надо поместить в BH номер страницы (всегда 0 для монохромного дисплея).

```
;---определение позиции курсора
MOV  AH,3           ;номер функции
MOV  BH,0           ;страница 0
INT  10H            ;строка:столбец в DH:DL
```

MS-DOS предоставляет две Esc-последовательности для сохранения и восстановления позиции курсора. Это специальные строки, которые, если их "вывести" на терминал, управляют монитором. Основы использования этих последовательностей описаны в приложении Д. Последовательность для запоминания позиции курсора - Esc[s, а для восстановления - Esc[u. Нет нужды запоминать координаты в переменной.

```
;---в сегменте данных
SAV_CUR DB   27,'[s$'       ;сохранение позиции курсора
RES_CUR DB   27,'[u$'       ;восстановление позиции курсора
;---сохранение курсора
        LEA  DX,SAV_CUR     ;адрес начала строки в DX
```

```
        MOV AH,9            ;номер функции вывода строки
        INT 21H             ;сохраняем позицию курсора
;---восстановление курсора
        LEA DX,RES_CUR      ;адрес начала строки в DX
        MOV AH,9            ;номер функции вывода строки
        INT 21H             ;восстанавливаем позицию курсора
```

Низкий уровень

Регистры 14 и 15 микросхемы 6845 хранят текущую позицию курсора, как объяснялось в [4.1.1]. Старший байт хранится в регистре 14. Два байта хранят числа от 0 до 1999 в режиме 80 символов в строке и от 0 до 999 в режиме 40 символов. Вам необходимо перевести получаемое число в координаты строки и столбца. Вы можете прочитать это значение, чтобы узнать текущую позицию видимого курсора на экране. Но запоминание этого значения и последующее восстановление его в регистрах не обязательно приведет к возврату курсора в предыдущую позицию, особенно если ваша программа использует любую из обычных функций работы с экраном, предоставляемых операционной системой. Это происходит потому, что BIOS хранит положение курсора в своих переменных, для того чтобы иметь возможность управлять страницами дисплея [4.5.3]. После того как вы восстановите регистры 14 и 15, курсор переместится в соответствующую позицию, но при следующем вызове прерывания вывода на экран курсор вернется назад к той позиции, в которой он должен находиться согласно значениям переменных BIOS.

4.2.6. Создание альтернативных типов курсора

Все прерывания операционной системы, связанные с выводом на экран, используют курсор. Вы можете изменить форму курсора с помощью техники, показанной в [4.2.4], или сделать курсор невидимым [4.2.3]. Возможны альтернативные типы курсора, когда вывод на экран осуществляется с помощью метода прямого отображения в память [4.3.1]. При этом "истинный" курсор выключается, поскольку он не будет адресовать символы в определенную позицию видеобуфера. Вместо этого создается "фальшивый" курсор с помощью байта атрибутов.

Наиболее эффективным методом является установка атрибута вывода в негативе для символа, на который указывает курсор. Для черно-белого экрана в качестве этого атрибута следует применять код ASCII 112. Другой способ - заставить символ, на который ука-

зывает курсор, мигать. В этом случае надо просто добавить 128 к текущему значению атрибута, чтобы символ начал мигать, и вычесть 128, чтобы прекратить мигание. Третий способ - установить для символа режим подчеркивания (код ASCII 1). И наконец, в программах, использующих командную строку, можно рассмотреть применение специального графического символа, который следует за последним символом командной строки, такого, как стрелки, выводимые кодами ASCII 17 или 27. Отметим, что когда программа получает ввод в нескольких режимах, вы можете помочь идентифицировать текущий режим за счет особого типа курсора.

Высокий уровень

В данном примере курсор формируется за счет вывода символа в позиции курсора в негативе. Переменная CURSORPOSITION хранит смещение символа, на который указывает курсор в видеобуфере. Это четное число в интервале от 0 до 3998. Прибавление к этой переменной 1 дает позицию байта атрибутов для этого символа, и, поместив туда 112, мы обеспечим вывод этого символа в негативе. Переменная FORMERATTRIBUTE хранит обычные атрибуты символа, с тем чтобы можно было восстановить их после того, как курсор сдвинется.

```
500 '''процедура анализа поступающих расширенных кодов
.
560 IF EXTCOD = 77 THEN GOSUB 5000'курсор вправо
.
5000 '''процедура, сдвигающая курсор вправо на одну позицию
5010 POKE CURPOS + 1,FORMERATR      'восстанавливаем атрибут
5020 CURPOS = CURPOS + 2            'новая позиция
5030 FORMERAT = PEEK(CURPOS + 1)    'сохраняем атрибут
5040 POKE CURPOS + 1,112            'включаем негатив
5050 RETURN                         'все сделано
```

Низкий уровень

Здесь тот же пример реализован на ассемблере:

```
;---процедура перемещения курсора на одну позицию вправо
CUR_RIT: MOV  BX,CURPOS          ;получение позиции
         INC  BX                 ;указываем на атрибут символа
```

```
MOV  AL,FORMERAT      ;берем сохраненный атрибут
MOV  ES:[BX],AL       ;восстанавливаем его
INC  BX               ;указываем на следующий символ
MOV  CURPOS,BX        ;сохраняем его смещение
MOV  AL,ES:[BX] + 1   ;получаем атрибут нового символа
MOV  FORMERAT,AL      ;сохраняем его
MOV  AL,112           ;помещаем атрибут вывода в негативе
MOV  ES:[BX] + 1,AL   ;засылаем его для следующего символа
```

4.3. Вывод символов на экран

Существует много способов вывода символов на экран. Иногда просто помещают один символ, белый на черном, в текущую позицию курсора. Другие методы более сложны, но дают больше возможностей управления размещением символов, а также их атрибутами и цветами. Некоторые процедуры выводят на экран целые строки. Но в любом случае базовой операцией, на которой основан вывод, является помещение кода ASCII выводимого символа в указанную позицию видеобуфера; при этом может также записываться и байт атрибутов в следующий адрес памяти.

Ваши программы могут помещать эти коды непосредственно в буфер. Этот метод называется отображением в память. Отображение в память, как правило, при программировании для выполнения заданной функции требует больше усилий, чем при использовании процедур операционной системы, но в результате получают более быстрый вывод на экран. Фирма IBM не рекомендует применять этот метод вывода на экран, поскольку будущие изменения аппаратуры могут привести к тому, что программы будут работать неверно. Но пока все новые разработки IBM следуют одной и той же схеме адресации, на которой основано отображение в память.

4.3.1. Вывод на экран одного символа

Все процедуры для вывода символа на экран в BIOS и DOS (а также в Бейсике) помещают символ в текущую позицию курсора и автоматически передвигают курсор на одну позицию вправо. Все они переносят вывод на следующую строку при достижении конца строки, если не сделано специальных указаний отбрасывать все символы за 80-м столбцом [4.2.2]. Важное различие между отдель-

ными процедурами состоит в том, что некоторые из них вместе с символом пишут также и его атрибуты, а некоторые этого не делают.

Рис. 4.2. Отображение в память для монохромного адаптера

Как в языках высокого, так и в языках низкого уровня символы могут выводиться на экран без обычных операций печати. Вместо этого используется прямое отображение в память, при котором коды символов и их атрибуты прямо засылаются в ячейки памяти видеобуфера, соответствующие определенной позиции курсора на экране. Буфер начинается с адреса B000:0000 для монохромного адаптера и с адреса B800:0000 для цветного графического адаптера и PCjr. EGA использует те же адреса в аналогичных режимах экрана. Позиции с четными номерами (начиная с нуля) содержат коды ASCII символов, а позиции с нечетными номерами - байты атрибутов. На рис. 4.2 показан участок памяти видео-

буфера. При этих операциях позиция курсора не меняется, и при желании он может быть выключен [4.2.3]. Вместо курсора надо хранить переменные, служащие указателями на текущую позицию.

Высокий уровень

Бейсик выводит как отдельные символы, так и целые строки с помощью одних и тех же операторов PRINT и WRITE. Как правило, используется PRINT; WRITE - это один из вариантов со специальными форматами вывода. PRINT работает с данными трех видов. Он выводит содержимое как строковых, так и числовых переменных, например PRINT S$ или PRINT X. Он выводит также символы, вставленные (в кавычках) внутрь самого оператора PRINT, например PRINT "This words are printed", и символы, соответствующие кодам ASCII, включенным в оператор PRINT в виде операторов CHR$, например PRINT CHR$(65), что обеспечивает вывод на экран символа A (код ASCII #65).

В одном операторе PRINT может выводиться много данных, при этом все три формы данных могут быть перемешаны. Отдельные данные ограничиваются запятой или точкой с запятой. Запятая приводит к тому, что следующие данные будут выводиться со следующей позиции табуляции данной строки. Точка с запятой приводит к тому, что данные печатаются на экране подряд, не разделенные пробелами (отметим, что PRINT вставляет пробел перед выводом любой числовой переменной, а WRITE не делает этого). Обычно оператор PRINT автоматически делает перевод на новую строку при завершении, таким образом следующий такой оператор начнет вывод с новой строки экрана. Чтобы перенос на новую строку не происходил, надо в конце оператора PRINT поставить точку с запятой, например PRINT S$;.

Для установки позиции курсора перед выводом используется оператор LOCATE. Без оператора LOCATE PRINT всегда начинает вывод с первой позиции строки, в которой находится курсор. Последовательные операторы PRINT заполняют экран до тех пор, пока не будет записана 24-я строка, после чего экран сдвигается вверх, с тем чтобы следующий оператор PRINT снова выводил 24-ю строку. PRINT может осуществлять вывод в 25-й строке только при помощи LOCATE; и это также приводит к автоматическому сдвигу экрана вверх. Чтобы запретить сдвиг, надо окончить оператор PRINT точкой с запятой. Однако этот метод не сработает в последних позициях строк 24 и 25. Для заполнения этих позиций без сдвига экрана нужно использовать отображение в память, как показано ниже.

Вы можете включать управляющие символы [7.1.9] внутрь оператора PRINT для того, чтобы реализовать перемещения курсора внутри строки. Например, если вы поместите в строку CHR$(13), то в этой точке будет сделан возврат каретки. Если вы выведете оператором PRINT строку "One" + CHR$(13) + "Two" + CHR$(13) + "Three", то в результате каждое слово будет выводиться с новой строки. Коды ASCII 28-31 сдвигают курсор на одну позицию соответственно вправо, влево, вверх и вниз. Оператор PRINT, не содержащий данных, приводит к выводу возврата каретки и, таким образом, следующий оператор PRINT будет делать вывод в строке через одну.

Прямое отображение в память существенно увеличивает скорость вывода на экран в Бейсике. Оно особенно полезно при конструировании табличного вывода, когда формы могут достигать правого нижнего угла экрана. Сначала надо установить указатель сегмента на &HB000, а затем использовать оператор POKE для засылки байтов памяти. Прилегающие по горизонтали символы отстоят друг от друга на два байта, разделяемые байтом атрибутов. Для 80-символьных экранов прилегающие по вертикали символы отстоят на 160 байт друг от друга (2 байта для каждого символа и атрибутов). В следующих двух примерах вдоль границы экрана рисуется рамка с помощью символов псевдографики. В первом примере чаще применяется оператор PRINT, а во втором используется исключительно прямое отображение в память. Отметим, что и в первом случае приходится использовать прямое отображение в память в последних столбцах строк 24 и 25, чтобы избежать сдвига экрана.

Использование PRINT:

```
10 CLS: KEY OFF                          'очистка экрана
20 DEF SEG = &HB000                      'указываем на видеобуфер
30 LOCATE 1,1: PRINT CHR$(201)           'левый верхний угол
40 LOCATE 1,80: PRINT CHR$(187)          'правый верхний угол
50 LOCATE 1,24: PRINT CHR$(186)
60 LOCATE 1,25: PRINT CHR$(200)
70 POKE 3838,186                         'позиция 80 строки 24
80 POKE 3998,188                         'позиция 80 строки 25
90 FOR N = 2 TO 79                       'горизонтальные линии
100 LOCATE 1,N: PRINT CHR$(205);: LOCATE 25,N: PRINT CHR$(205)
110 NEXT
120 FOR N = 2 TO 23                      'вертикальные линии
130 LOCATE N,1: PRINT CHR$(186): LOCATE N,80: PRINT CHR$(186)
140 NEXT
```

Использование прямого отображения в память:

```
10 CLS: KEY OFF                          'очистка экрана
20 DEF SEG = &HB000                      'буфер монохромного дисплея
30 POKE 0,201                            'левый верхний угол
40 POKE 158,187                          'правый верхний угол
50 POKE 3840,200                         'левый нижний угол
60 POKE 3998,188                         'правый нижний угол
70 FOR N = 2 TO 156 STEP 2               'горизонтальные прямые
80 POKE N,205: POKE N + 3840,205         'как верхняя, так и нижняя
90 NEXT
100 FOR N = 160 TO 3680 STEP 160         'вертикальные прямые
110 POKE N,186: POKE N + 158,186         'правая и левая
120 NEXT
```

Средний уровень

Операционная система предоставляет шесть процедур вывода на экран - три в BIOS и три в DOS. Они различаются главным образом тем, передвигается ли курсор после вывода символа, вызывают ли они сдвиг экрана, позволяют ли они устанавливать атрибуты и цвета символов, а также какие управляющие коды они интерпретируют (некоторые рассматривают символ «Backspace» просто как обычный символ, а некоторые действительно сдвигают курсор на одну позицию назад). Эти шесть процедур следующие:

Прерывание 10H:

функция	9	вывод символа с атрибутами
	A	вывод символа без атрибутов
	E	"телетайпная" процедура (как на принтер)

Прерывание 21H:

функция	2	вывод символа без атрибутов
	6	вывод символа без атрибутов
	9	вывод строки символов

Функции 9 и A прерывания 10H вообще не интерпретируют управляющие символы. Функции DOS интерпретируют управ-

ляющие коды, приведенные в следующей таблице. Функция E прерывания 10H интерпретирует все коды таблицы, кроме ASCII 9.

ASCII 7	звонок
ASCII 8	возврат на шаг (Backspace)
ASCII 9	табуляция
ASCII 10	перевод строки
ASCII 13	возврат каретки

Первые две функции прерывания 10H не передвигают курсор после вывода символа. Функция 9 этого прерывания выводит на экран с указанием атрибутов, а функция A - без указания, при этом сохраняется текущее значение байта атрибутов для этого символа. AL должен содержать выводимый символ, а BL - атрибуты. Номер страницы дисплея содержится в BH. Он должен указываться даже для монохромного дисплея, который имеет только одну страницу памяти дисплея. В этом случае должна быть установлена первая страница, которой соответствует номер 0. Особое свойство этих двух функций BIOS состоит в том, что символ выводится такое число раз, какое указано в CX. Обычно указывают CX равным 1, но эти функции могут легко выводить целые строки символов, если указать большее значение счетчика, - полезное свойство при создании рамок. Отметим, что даже если выводится много символов, позиция курсора не изменяется. Когда строка выводимых символов займет все свободное пространство экрана справа-вниз от курсора, вывод будет перенесен в первые позиции экрана.

```
;---вывод символа в негативе
MOV  AH,9              ;функция записи с атрибутами
MOV  AL,THE_CHAR       ;символ в AL
MOV  BL,112            ;атрибуты в BL
MOV  BH,0              ;страница 1
MOV  CX,1              ;вывести один раз
INT  10H
```

Вместо того чтобы постоянно восстанавливать значение счетчика в CX, прерывание BIOS предоставляет телетайпную процедуру, которая больше подходит для вывода строки символов. Эта процедура выполняется с помощью функции E. Подготовка к ней

такая же, как и для функции A, но не надо засылать значение в CX. Строка выводится за счет изменения символа в AL и повторного вызова прерывания. При использовании в графическом режиме в BL устанавливается цвет палитры, в противном случае сохраняется старый атрибут.

```
;---вывод строки с помощью телетайпной процедуры
        MOV AH,0EH          ;номер функции
        MOV BH,0            ;номер страницы
        LEA BX,STRING       ;BX указывает на строку
NEXT:   MOV AL,[BX]         ;берем символ в AL
        CMP AL,'$'          ;проверка на конец строки
        JE  ALL_DONE        ;если да, то выход
        INT 10H             ;вывод строки
        INC BX              ;переходим к следующему символу
        JMP SHORT NEXT      ;повторяем процедуру
ALL_DONE:
```

Прерывание DOS 21H, как правило, предоставляет более полезные процедуры, перемещающие курсор и приводящие к сдвигу экрана при достижении нижней строки, а также интерпретирующие некоторые из обычных управляющих кодов. Функции DOS осуществляют вывод на страницу, которая должна быть установлена функцией 5 прерывания 10H [4.5.3]. Предоставляются две функции для вывода символа - с номерами 2 и 6. Первая из них распознает «Ctrl-Break» [3.2.8], а вторая нет. (Когда с клавиатуры вводится «Ctrl-Break», процедура обработки «Ctrl-Break» не выполняется до тех пор, пока не используется функция, которая распознает его наличие.)

Обе функции выводят белые символы на черном фоне до тех пор, пока не сделана специальная установка цвета с помощью драйвера устройства ANSI.SYS [4.1.3]. В общем, необходимо только поместить символ в DL, номер функции - в AH и вызвать прерывание 21H. Однако функция 6 имеет и второе назначение - ввод с клавиатуры. Она выступает в этой роли, только если в DL помещен код FF [3.1.5]. Во всех остальных случаях она выводит на экран содержимое DL. В следующем примере функция 6 поочередно принимает и печатает символ (в [3.1.4] обсуждается процедура, которая комбинирует оба этих свойства).

```
        MOV AH,6            ;номер функции
NEXT:   MOV DL,0FFH         ;при этом значении принимаем ввод
```

```
INT   21H          ;выполняем прерывание
JZ    NEXT         ;если не было ввода, то обратно
CMP   AL,13        ;это был возврат каретки?
JE    END_INPUT    ;если да, то на конец
MOV   DL,AL        ;иначе,посылаем символ в DL
INT   21H          ;и выводим его на экран
JMP   SHORT NEXT   ;повторяем процедуру
```

Низкий уровень

На нижнем уровне весь вывод на экран осуществляется через отображение в память. Такой метод не рекомендуют, чтобы не столкнуться с проблемой совместимости с будущими поколениями машин. Однако видеобуфер микрокомпьютеров фирмы IBM устроен одинаково и расположен в одних и тех же адресах памяти. Поскольку буфер устроен таким образом, что байты атрибутов перемежаются с байтами символов, символьные данные не могут просто пересылаться из памяти в буфер инструкцией MOVSB, поскольку указатель в буфере должен увеличиваться на два после каждого переноса байта. Однако такой способ существенно ускоряет вывод на экран. Отметим, что отображение в память не работает при выводе символов в графическом режиме. В этом случае размер видеобуфера 16K или 32K и BIOS рисует каждый символ поточечно. Отметим также, что при отображении в память не используется курсор для указания на символ. При желании можно перемещать курсор по мере ввода [4.2.1] или выключить его и создать свой псевдокурсор [4.2.6].

```
;---в сегменте данных
SAM_STR DB   'PRINT THIS STRING$'
;---вывод строки
        MOV AX,0B000H   ;монохромный дисплей
        MOV ES,AX       ;указываем на видеобуфер
        LEA BX,SAM_STR  ;BX указывает на строку
        MOV DI,CUR_ST   ;начальная позиция в буфере
NEXT:   MOV AL,[BX]     ;берем символ
        CMP AL,'$'      ;проверка на конец строки
        JE  ALL_DONE    ;если да, то выход
        MOV ES:[DI],AL  ;иначе - помещаем символ в буфер
        INC DI          ;увеличиваем указатель на 2
        INC DI
```

```
        INC  BX             ;переходим к обработке следующего символа
        JMP  SHORT NEXT
ALL_DONE:
```

Для цветного графического адаптера и PCjr (но не для EGA) существует проблема, связанная с отображением в память. Когда запись в буферную память происходит одновременно с чтением ее для вывода на экран, на экране возникает интерференция. Эта проблема решается ожиданием сигнала "все чисто" (all clear) перед записью в видеобуфер. Надо непрерывно читать значение из порта 3DAH. Когда бит 0 равен 1, можно спокойно писать. (3DAH - это порт, через который PCjr посылает данные массиву ворот дисплея; когда из него читаем, он возвращает регистр статуса, как и для цветного адаптера.)

```
;---ожидаем пока все чисто
        MOV DX,3DAH         ;порт регистра статуса
CH_AGN: IN  AL,DX           ;получаем значение
        TEST AL,1           ;проверка первого бита
        JNE  CH_AGN         ;если он 0, то обратно
;---теперь выводим сообщение
        LEA  BX,MESS        ;сообщение в сегменте данных
        MOV  DI,2000        ;начинаем вывод с центра экрана
        MOV  AH,01000001B   ;атрибут синий на красном
NEXT:   MOV  AL,[BX]        ;берем символ
        CMP  AL,'$'         ;проверяем на конец строки
        JE   ALL_DONE       ;если конец, то на выход
        MOV  ES:[DI],AX     ;иначе - выводим символ
        INC  BX             ;увеличиваем указатель строки
        INC  DI             ;увеличиваем указатель буфера
        INC  DI
        JMP  SHORT NEXT     ;обрабатываем следующий символ
ALL_DONE:
```

Попробуйте поэкспериментировать, сколько символов за один цикл может выводить ваша процедура без появления интерференции. Имейте в виду, что при первом выполнении цикла тестируемый бит может быть равным единице. Однако может не остаться времени, чтобы завершить операцию записи.

PCjr специально сконструирован таким образом, что вывод в адреса, используемые буфером цветного графического дисплея,

перенаправляется в ту область памяти, где находится буфер. Это свойство позволяет создавать программное обеспечение, подходящее для обеих систем.

4.3.2. Вывод строки символов на экран

Процедуры, которые выводят целые строки символов, очень полезны, но они могут накладывать ограничения на содержимое выводимой строки. Надо обращать внимание на то, какие управляющие коды (табуляция, пробел и т.п.) интерпретируются, а какие нет. До появления AT BIOS не имел функции вывода строки, хотя в MS-DOS всегда существовала такая функция. Функция BIOS предоставляет больший контроль над атрибутами символов. Естественно, что ее применение создает проблему совместимости с предыдущими машинами. Напоминаем, что EGA имеет ПЗУ, расширяющее ROM-BIOS, и функция вывода строки символов является одним из таких расширений. В этом случае любой IBM PC и XT имеет возможность использовать эту процедуру.

Высокий уровень

Бейсик выводит строку точно так же, как и отдельные символы. Надо просто написать PRINT S$, где S$ может быть любой строкой длиной до 255 символов, которую сконструировала программа. Интерпретируются 10 управляющих кодов, а именно:

ASCII 7	звонок
ASCII 9	табуляция
ASCII 10	перевод строки
ASCII 11	курсор в первую позицию экрана (Home)
ASCII 12	перевод формата (стирает экран + Home)
ASCII 13	возврат каретки
ASCII 28	курсор вправо
ASCII 29	курсор влево
ASCII 30	курсор вверх
ASCII 31	курсор вниз

Все остальные коды выводятся на экран как символы.

Средний уровень

Функция 9 прерывания 21H выводит строку. DS:DX должны указывать на первый символ строки. Строка должна завершаться символом $, что означает, что сам символ $ не может входить в строку. Строка может быть любой длины. Функция не переводит автоматически курсор на начало следующей строки после завершения вывода; чтобы это выполнялось, надо добавить в конец строки символы 0AH (перевод строки) и 0DH (возврат каретки).

```
;---в сегменте данных
FIR_STR DB    'This is the first string',0AH,0DH,'$'
SEC_STRDB     'And this is the second string$'
;---вывод строки
        MOV   AH,9          ;номер функции вывода строки
        LEA   DX,FIR_STR    ;загружаем адрес первой строки
        INT   21H           ;печатаем строку с позиции курсора
        LEA   DX,SEC_STR    ;загружаем адрес второй строки
        INT   21H           ;печатаем строку с начала новой строки
```

Интерпретируются следующие управляющие коды:

ASCII	7	звонок
ASCII	8	возврат на шаг (Backspace)
ASCII	9	табуляция
ASCII	10	перевод строки
ASCII	13	возврат каретки

Функция DOS 40H прерывания 21H также полезна при выводе строк на экран. Она требует, чтобы вы знали длину строки, поскольку она не нуждается в символе-ограничителе; эта функция особенно удобна для дампа текстовых файлов на экран. Исходно эта функция была предназначена для вывода в файл. Она требует дескриптора, который является идентификационным номером для данного файла или устройства. Дисплей имеет заранее предназначенный дескриптор #1. Надо поместить дескриптор в BX, а число байтов строки - в CX. DS:DX должны указывать на строку. Функция выводит текст с нормальными (белый на черном) атрибутами. Отметим, что не надо предварительно "открывать" дисплей, как это вы делаете с другими файлами при использовании этой функции. Вот пример:

```
;---вывод 1000 байт текста
MOV  AH,40H          ;номер функции
MOV  BX,1            ;дескриптор дисплея
LEA  DX,STRING       ;загружаем адрес строки
MOV  CX,1000         ;число выводимых байтов
INT  21H             ;
```

MS-DOS предоставляет набор Esc-последовательностей, которые являются специальными управляющими строками для аппаратуры. Когда они выводятся с помощью функции 9 прерывания 21H, они могут управлять курсором, режимом дисплея, цветом символов и некоторыми аспектами клавиатуры. В приложении Д обсуждается, как их использовать. Когда программа выводит на экран много строк, Esc-последовательности часто являются самым удобным способом позиционирования курсора и установки цвета строки. Это происходит потому, что они сами рассматриваются как очередные строки в серии выводимых строк.

У AT и машин, снабженных EGA, функция 13H прерывания 10H выводит строку. ES:BP должны указывать на строку, а длина строки должна быть в CX. DX указывает позицию курсора, откуда будет начинаться строка (вычисляемую как смещение от начала страницы, на которую идет вывод без учета байтов атрибутов). В BX должен быть указан номер страницы. Наконец, номер кода от 0 до 3, содержащийся в AL, указывает способ вывода строки.

AL = 0	строка состоит только из символов, курсор неподвижен
AL = 1	строка состоит только из символов, курсор движется
AL = 2	в строке чередуются символы и атрибуты, курсор неподвижен
AL = 3	в строке чередуются символы и атрибуты, курсор движется

Когда AL равен 0 или 1, атрибуты должны находиться в BL. Все символы будут выводиться с этими атрибутами. Данная функция интерпретирует возврат на шаг, перевод строки, возврат каретки и звонок как управляющие команды, а не как печатаемые символы.

Низкий уровень

Ограничение на использование символа $ делает функцию 9 бесполезной для большинства приложений. Однако на многих машинах это единственное прерывание, доступное для вывода строки неизвестной длины. Попробуйте написать свое собственное прерывание (в [1.2.3] показано как), использующее технику отображения в память [4.3.1]. Возьмите в качестве ограничителя

какой-нибудь специальный символ, например ASCII 0, вместо $. Пусть эта процедура обрабатывает только те управляющие коды, которые нужны вам. Такой метод будет работать намного быстрее, чем при использовании функции MS-DOS.

4.3.3. Чтение символа и его атрибутов в данной позиции

Обычно программа получает данные из своих переменных и помещает их в видеобуфер для вывода на экран. В некотором смысле программа "знает", что на экране. Но встречаются ситуации, в которых сам видеобуфер используется как рабочая область (например, в графических программах вырезки и вставки) и текущее содержимое экрана не записано в памяти программы. В этих случаях иногда необходимо прочитать с экрана, вместо того чтобы вывести на него. Функция BIOS позволяет прочитать символ и его атрибуты в определенной позиции экрана; другой способ состоит в обращении метода прямого отображения в память дисплея [4.3.1]. Чтобы прочитать символ и атрибуты в строке 0 и столбце 39 (1,40 в Бейсике) в режиме 80 символов в строке, надо сложить (0 x 160) плюс (39 x 2) и взять результат в качестве смещения в видеобуфере. Смещения для различных страниц обсуждаются в [4.5.3]. Имейте в виду, что обращение метода прямого отображения в память не будет работать при выводе символов в графическом режиме.

Высокий уровень

В Бейсике функция SCREEN служит для получения символа или атрибутов (эта функция не имеет ничего общего с оператором SCREEN, устанавливающим режим дисплея). SCREEN 5,10 получает код ASCII символа, расположенного в строке 5, столбце 10 (строки и столбцы нумеруются от 1). Чтобы получить атрибуты символа, надо добавить третий параметр 1, например SCREEN 5,10,1. В графическом режиме данная функция возвращает 0, если требуемая позиция экрана не содержит (немодифицированного) символа.

Атрибуты также возвращаются в виде кода от 0 до 255. Поскольку Бейсик не позволяет использования двоичных чисел, требуются некоторые манипуляции, чтобы определить атрибуты. Основной цвет равен ATTRIBUTE MOD 16. После того как вы выделили основной цвет, цвет фона определяется по формуле (((ATTRIBUTE - FOREGROUND)/16) MOD 128). Если байт атрибутов больше 127, то включено мигание (или, при соответ-

ствующей установке, включены интенсивные цвета фона [4.1.3]). В приложении Б обсуждаются битовые операции в Бейсике.

Средний уровень

Функция 8 прерывания 10H возвращает символ и его атрибуты для текущей позиции курсора. В BH должен содержаться номер текущей страницы дисплея (отсчитываемый от 0 и всегда равный 0 для монохромного дисплея). Код символа возвращается в AL, а байт атрибутов в AH. Эта функция настолько мощная, что способна даже читать символы в графическом режиме, сообщая цвет палитры в AH. Она работает даже для символов, определяемых пользователем [4.3.4]. В примере определяются символ и атрибуты в позиции 0,39 для страницы 2 графического адаптера:

```
;---установка позиции курсора
MOV  AH,2          ;функция установки курсора
MOV  DH,0          ;номер строки
MOV  DL,39         ;номер столбца
MOV  BH,0          ;номер страницы
INT  10H           ;позиционируем курсор
;---чтение символа и атрибутов
MOV  AH,8          ;функция чтения символа/атрибутов
MOV  BH,2          ;номер страницы
INT  10H           ;в AH:AL теперь атрибуты и символ
```

Низкий уровень

Надо вычислить смещение и проделать операцию, обратную прямой записи в память. При необходимости надо добавить смещение для данной страницы. В примере получаем символ и атрибуты в позиции 7,39 страницы 2 графического адаптера:

```
;---чтение символа и атрибутов позиции 7,39 страницы 2
MOV  AX,0B800H     ;адрес видеобуфера
MOV  ES,AX         ;ES указывает на первый байт буфера
MOV  DI,1000H      ;смещение до начала страницы
MOV  AL,80         ;умножаем номер строки на 160
MOV  BL,7          ;номер строки
MUL  BL            ;теперь в AX (строка-1)x160
MOV  AX,39         ;номер столбца
```

```
ADD  BX,AX              ;номер позиции в видеобуфере
SHL  BX,1               ;умножаем его на два
MOV  AX,ES:[BX][DI]     ;теперь AH:AL содержат атрибуты/символ
```

4.3.4. Создание специальных символов

Только монохромный адаптер не может выводить символы вида, заданного самим программистом. Цветной адаптер поддерживает 128 символов, определяемых пользователем, PCjr - 256, а EGA - 1024, из которых одновременно доступно 512. Для цветного адаптера ROM-BIOS содержит данные для разрисовки только первых 128 символов набора ASCII (с номерами от 0 до 127). Следующие 128 символов недоступны для вас, пока вы не создадите их, используя описанную здесь технику. Отметим, что в MS-DOS 3.00 имеется команда GRAFTABL, которая предоставляет требуемые данные для второй порции из 128 символов. PCjr имеет готовые данные для второй порции из 128 символов; а EGA - полные наборы символов для режимов с 200 и с 350 строками.

Символы для графического адаптера и PCjr описываются с помощью матрицы 8*8 точек. Данные для каждого символа содержатся в восьми байтах. Каждый байт содержит установку для точек одного ряда, начиная с верхнего, причем старший бит (номер 7) соответствует самой левой точке в ряду. Когда соответствующий бит равен 1, точка высвечивается. Для описания символа вы должны определить правильные последовательности битов для восьми байтов и поместить их в последовательные ячейки памяти. На рис. 4.3 показано, как 8 байт описывают бубновую масть.

Все 128 символов вместе требуют 1024 байта, хотя вовсе не обязательно, чтобы были описаны все символы. Специальный вектор прерывания (постоянный указатель в младших адресах памяти [1.2.0]) указывает адрес первого байта первого символа расширенного набора, т.е. символ номер 128. Когда в позицию символа в видеобуфере посылается код 128, просматриваются и выводятся первые восемь байт. Если номер символа 129, то выводятся байты с девятого по шестнадцатый и т.д.

Номер этого вектора прерывания 1FH и он расположен по адресу 0000:007C. Поместите значение смещения в младшее слово (сначала младший байт), а адрес сегмента - в старшее слово (снова сначала младший байт). Отметим, что можно определять символы с бо́льшими номерами кодов, не отводя памяти для символов с меньшими номерами; надо просто, чтобы вектор указывал на некоторый адрес, который меньше, чем адрес начала блока, содержащего данные для описания символов. Восьмибайтовые

последовательности, описывающие символы ASCII с кодами 128-255, приведены в [4.3.5]. У PCjr вектор 1FH указывает на вторые 128 символов ASCII, а вектор 44H - на первые. Оба этих вектора можно изменить, допуская полный набор 256 символов, определяемых пользователем.

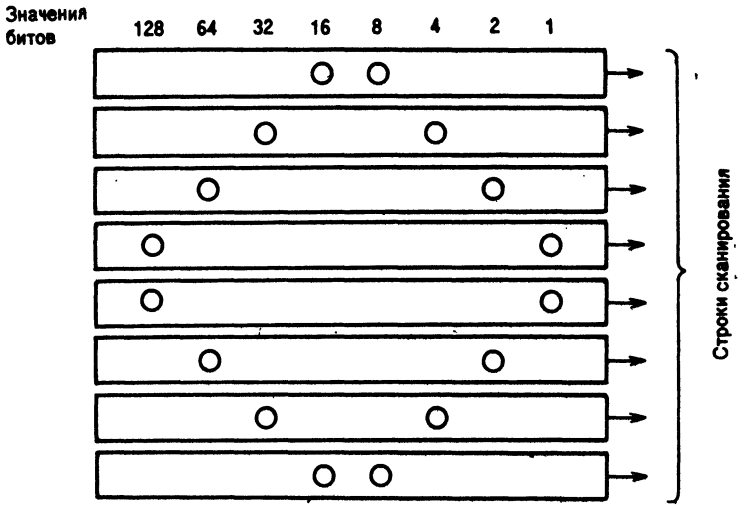


Рис. 4.3. Цепочка битов, образующая бубновую масть

Для EGA картина намного сложнее, хотя и более гибкая. При инициализации текстового режима один из двух наборов символов (8*8 или 8*14) копируется из ПЗУ EGA в карту битов 2 видеобуфера. Эта часть буфера рассматривается как разбитая на блоки, причем стандартный набор символов помещается в блок 0. При условии, что EGA оснащен достаточной памятью, могут быть определены еще три блока для описания символов. Размер блока определяется числом строк матрицы, используемой для описания символа. Символы, описываемые матрицей 8*8, требуют 8 x 256, или 2048 байт. Когда разрешено более одного блока символов, бит

3 байта атрибутов определяет, из какого блока будут браться данные для описания символа.

Какой из блоков будет необходим - зависит от установки битов 0-3 регистра выбора карты символов, адрес порта которого 3C5H. Предварительно надо послать 3 в порт 3C4H, чтобы указать требуемый регистр. Биты 1-0 дают номер блока символов, который берется, когда бит 3 байта атрибутов равен 0, а биты 3-2 - делают то же самое, когда бит 3 равен 1. Когда установка обеих пар битов совпадает, возможность использования двух наборов символов отсутствует и бит 3 байта атрибутов переключается на установку интенсивности символа. В этом случае используется только блок 0. Однако ничто не может помешать вам поместить свои символы в любую нужную вам позицию в этом блоке. Если вы изменили стандартный набор символов, то можно в любой момент восстановить его из ПЗУ.

Высокий уровень

В Бейсике вы должны позаботиться о том, чтобы данные, описывающие символы, находились за пределами памяти, используемой программой. Если памяти много, то можно поместить данные в старшие адреса; если имеется опасность конфликта, то следует применить команду CLEAR для ограничения количества памяти, которую может использовать Бейсик. Затем нужно поместить адрес первого байта данных в вектор прерывания. В следующем примере описывается символ 128 как квадратная рамка. Операторы DATA содержат значения, описывающие символ. Они равны либо 255, либо 129; в первом случае все биты равны 1, а во втором равны 1 только крайние биты. Вычисление десятичных значений, соответствующих данным цепочкам битов, рассмотрено в приложении Б.

```
100 '''помещаем данные, начиная с адреса &H3000
110 DATA 255, 129, 129, 129, 129, 129, 129, 255'данные для одного символа
120 DEF SEG = &H3000                          'указываем начало сегмента
130 FOR N = 0 TO 7                            'определяем 8 байт
140 READ Q                                    'читаем 1 байт
150 POKE N,Q                                  'помещаем его в память
160 NEXT                                      'и т.д.
170 '''установка вектора прерывания
180 DEF SEG = 0                               'указываем на начало памяти
190 POKE 124,0                                'указываем смещение
200 POKE 125,0
210 POKE 126,0                                'указываем сегмент
220 POKE 127,&H30
```

```
230 '''печатаем символ
240 LOCATE 12,12: PRINT CHR$(128)          'теперь есть символ 128
```

Средний уровень

Для цветного адаптера и PCjr для изменения вектора прерывания 1FH применяйте функцию 25H прерывания 21H. При входе DS:DX должны указывать на первый байт блока данных. Более подробное описание см. в [1.2.3]. В примере создаются два символа с номерами 128 и 129. Они являются зеркальными отображениями друг друга, а выведенные подряд образуют небольшой прямоугольник.

```
;---в сегменте данных
CHAR_DAT DB   11111111B, 10000000B, 10000000B, 10000000B
         DB   10000000B, 10000000B, 10000000B, 11111111B
         DB   11111111B, 00000001B, 00000001B, 00000001B
         DB   00000001B, 00000001B, 00000001B, 11111111B
;---установка вектора прерывания
         PUSH DS                   ;сохраняем DS
         LEA  DX,CHAR_DAT          ;смещение для данных в DX
         MOV  AX,SEG CHAR_DAT      ;сегмент для данных в DS
         MOV  DS,AX
         MOV  AH,25H               ;функция установки вектора
         MOV  AL,1FH               ;номер изменяемого вектора
         INT  21H                  ;установка вектора
         POP  DS                   ;восстанавливаем DS
;---печать символов
         MOV  AH,2                 ;номер функции
         MOV  DL,128               ;первый символ
         INT  21H                  ;вывод его
         MOV  DL,129               ;второй символ
         INT  21H                  ;вывод его
```

Для адаптера EGA характерна функция 11H прерывания 10H, которая манипулирует набором символов. Эта функция может быть очень сложной, когда она применяется для создания специальных режимов экрана, но ее основное назначение достаточно простое. Существуют четыре подфункции. Когда AL равен 0, данные, определяемые пользователем, переносятся из памяти в специальный блок символов. Когда AL равен 1 или 2, наборы данных для символов 8*14 и 8*8 соответственно копируются из ПЗУ в блок символов. Когда AL равен 3, функция устанавливает назначение блока в регистре выбора карты символов, как описано выше. В последнем случае надо просто поместить соответствующие

данные в BL и вызвать функцию. Для загрузки данных из ПЗУ поместите номер блока в BL и выполните функцию. Для загрузки своих данных надо, чтобы ES:BP указывали на них, число передаваемых символов должно быть в CX, смещение (номер символа) в блоке должно быть в DX, число байтов на символ - в BH, а номер блока - в BL. После этого вызывайте прерывание 10H. Вот пример:

```
;---устанавливаем 128 пользовательских символов в блоке 0
MOV  AX,SEG CHAR_DAT ;ES:BP должны указывать на данные
MOV  ES,AX
MOV  BP,OFFSET CHAR_DAT
MOV  CX,128          ;число символов
MOV  DX,128          ;начальное смещение
MOV  BL,0            ;номер блока
MOV  BH,8            ;матрица 8*8
MOV  AL,1            ;номер подфункции
MOV  AH,11H          ;номер функции
INT  10H             ;переносим данные
```

Вы можете получить информацию о статусе имеющегося набора символов с помощью подфункции 30H функции 11H прерывания 10H. У этой подфункции нет входных регистров, а при возврате в CX содержится число байтов, отведенных для каждого символа, а в DX - сколько рядов точек символ занимает на экране (оба этих параметра зависят как от размера символа, так и от вертикального разрешения экрана).

4.3.5. Сводка данных для описания символов

Ниже приведены 8-байтовые последовательности, необходимые для описания символов для цветного графического адаптера. Их использование объяснено в [4.3.4].

Код ASCII	Символ	Последовательность (шестнадцатиричная)
128	Ç	78 CC C0 CC 78 18 0C 78
129	ü	00 CC 00 CC CC CC 7E 00
130	é	1C 00 78 CC FC C0 78 00
131	â	7E C3 3C 06 3E 66 3F 00
132	ä	CC 00 78 0C 7C CC 7E 00
133	à	E0 00 78 0C 7C CC 7E 00
134	å	30 30 78 0C 7C CC 7E 00
135	ç	00 00 78 0C 7C CC 7E 00

Код ASCII	Символ	Последовательность (шестнадцатиричная)
136	ê	7E C3 3C 66 7E 60 3C 00
137	ë	CC 00 78 CC FC C0 78 00
138	è	E0 00 78 CC FC C0 78 00
139	ï	CC 00 70 30 30 30 78 00
140	î	7C C6 38 18 18 18 3C 00
141	ì	E0 00 70 30 30 30 78 00
142	Ä	C6 38 6C C6 FE C6 C6 00
143	Å	30 30 00 78 CC FC CC 00
144	É	1C 00 FC 60 78 60 FC 00
145	æ	00 00 7F 0C 7F CC 7F 00
146	Æ	3E 6C CC FE CC CC CE 00
147	ô	78 CC 00 78 CC CC 78 00
148	ö	00 CC 00 78 CC CC 78 00
149	ò	00 E0 00 78 CC CC 78 00
150	û	78 CC 00 CC CC CC 7E 00
151	ù	00 E0 00 CC CC CC 7E 00
152	ÿ	00 CC 00 CC CC 7C 0C F8
153	Ö	C3 18 3C 66 66 3C 18 00
154	Ü	CC 00 CC CC CC CC 78 00
155	¢	18 18 7E C0 C0 7E 18 18
156	£	38 6C 64 F0 60 E6 FC 00
157	¥	CC CC 78 FC 30 FC 30 30
158	₨	F8 CC CC FA C6 CF C6 C7
159	ƒ	0E 1B 18 3C 18 18 D8 70
160	á	1C 00 78 00 7C CC 7E 00
161	í	38 00 70 30 30 30 78 00
162	ó	00 1C 00 78 CC CC 78 00
163	ú	00 1C 00 CC CC CC 7E 00
164	ñ	00 F8 00 F8 CC CC CC 00
165	Ñ	FC 00 CC EC FC DC CC 00
166	ª	3C 6C 6C 3E 00 7E 00 00
167	º	38 6C 6C 38 00 7C 00 00

Код ASCII	Символ	Последовательность (шестнадцатиричная)
168	¿	30 00 30 60 C0 CC 78 00
169	⌐	00 00 00 FC C0 C0 00 00
170	¬	00 00 00 FC 0C 0C 00 00
171	½	C3 C6 CC DE 33 66 CC 0F
172	¼	C3 C6 CC DB 37 6F CF 03
173	¡	18 18 00 18 18 18 18 00
174	«	00 33 66 CC 66 33 00 00
175	»	00 CC 66 33 66 CC 00 00
176		22 88 22 88 22 88 22 88
177	▒	55 AA 55 AA 55 AA 55 AA
178	▓	DB 77 DB EE DB 77 DB EE
179	│	18 18 18 18 18 18 18 18
180	┤	18 18 18 18 F8 18 18 18
181	╡	18 18 F8 18 F8 18 18 18
182	╢	36 36 36 36 F6 36 36 36
183	╖	00 00 00 00 FE 36 36 36
184	╕	00 00 F8 18 F8 18 18 18
185	╣	36 36 F6 06 F6 36 36 36
186	║	36 36 36 36 36 36 36 36
187	╗	00 00 FE 06 F6 36 36 36
188	╝	36 36 F6 06 FE 00 00 00
189	╜	36 36 36 36 FE 00 00 00
190	╛	18 18 F8 18 F8 00 00 00
191	┐	00 00 00 00 F7 18 18 18
192	└	18 18 18 18 1F 00 00 00
193	┴	18 18 18 18 FF 00 00 00
194	┬	00 00 00 00 FF 18 18 18
195	├	18 18 18 18 1F 18 18 18
196	-	00 00 00 00 FF 00 00 00
197	┼	18 18 18 18 FF 18 18 18
198	╞	18 18 1F 18 1F 18 18 18
199	╟	36 36 36 36 37 36 36 36

Код ASCII	Символ	Последовательность (шестнадцатиричная)
200	╚	36 36 37 30 3F 00 00 00
201	╔	00 00 3F 30 37 36 36 36
202	╩	36 36 F7 00 FF 00 00 00
203	╦	00 00 FF 00 F7 36 36 36
204	╠	36 36 37 30 37 36 36 36
205	═	00 00 FF 00 FF 00 00 00
206	╬	36 36 F7 00 F7 36 36 36
207	╧	18 18 FF 00 FF 00 00 00
208	╨	36 36 36 36 FF 00 00 00
209	╤	00 00 FF 00 FF 18 18 18
210	╥	00 00 00 00 FF 36 36 36
211	╙	36 36 36 36 3F 00 00 00
212	╘	18 18 1F 18 1F 00 00 00
213	╒	00 00 1F 18 1F 18 18 18
214	╓	00 00 00 00 3F 36 36 36
215	╫	36 36 36 36 FF 36 36 36
216	┼	18 18 FF 18 FF 18 18 18
217	┘	18 18 18 18 F8 00 00 00
218	┌	00 00 00 00 1F 18 18 18
219	▉	FF FF FF FF FF FF FF FF
220	▄	00 00 00 00 FF FF FF FF
221	▌	F0 F0 F0 F0 F0 F0 F0 F0
222	▐	0F 0F 0F 0F 0F 0F 0F 0F
223	▀	FF FF FF FF 00 00 00 00
224	α	00 00 76 DC CB DC 76 00
225	ß	00 78 CC F8 CC F8 C0 C0
226	Γ	00 CC C0 C0 C0 C0 00 00
227	π	00 FE 6C 6C 6C 6C 6C 00
228	Σ	FC CC 60 30 60 CC FC 00
229	σ	00 00 7E D8 D8 D8 70 00
230	µ	00 66 66 66 66 7C 60 C0
231	τ	00 76 DC 18 18 18 18 00

Код ASCII	Символ	Последовательность (шестнадцатиричная)
232	Φ	FC 30 78 CC CC 78 30 FC
233	Θ	38 6C C6 FE C6 6C 38 00
234	Ω	38 6C C6 C6 6C 6C EE 00
235	δ	1C 30 18 7C CC CC 78 00
236	∞	00 00 7E DB DB 7E 00 00
237	φ	06 0C 7E DB DB 7E 60 C0
238	ε	38 60 C0 F8 C0 60 38 00
239	∩	78 CC CC CC CC CC CC 00
240	≡	00 FC 00 FC 00 FC 00 00
241	±	30 30 FC 30 30 00 FC 00
242	≥	60 30 18 30 60 00 FC 00
243	≤	18 30 60 30 18 00 FC 00
244	⌠	0E 1B 1B 18 18 18 18 18
245	⌡	18 18 18 18 18 D8 D8 70
246	÷	30 30 00 FC 00 30 30 00
247	≈	00 76 DC 00 76 DC 00 00
248	°	38 6C 6C 38 00 00 00 00
249	∙	00 00 00 18 18 00 00 00
250	·	00 00 00 00 18 00 00 00
251	√	0F 0C 0C 0C EC 6C 3C 1C
252	η	78 6C 6C 6C 6C 00 00 00
253	²	70 18 30 60 78 00 00 00
254	■	00 00 3C 3C 3C 3C 00 00
255	(blank FF)	00 00 00 00 00 00 00 00

4.4. Вывод точечной графики

Цветной графический адаптер имеет три графических режима, PCjr - шесть, а EGA - семь. Как устанавливать эти режимы, показано в [4.1.2]. Требования к размеру памяти существенно различаются для доступных режимов в зависимости от разрешения экрана и числа цветов. В своих улучшенных графических режимах EGA использует память дисплея совсем по-другому, чем остальные видеосистемы, но он точно эмулирует их методы работы с памятью в трех общих режимах.

Сначала рассмотрим цветной адаптер и систему PCjr. Два цвета (черный и белый) требуют только один бит памяти для каждой точки на экране. Четыре цвета занимают 2 бита, а 16 цветов - 4 (8-цветные режимы не используются, поскольку три бита, требующиеся для их представления, нельзя удобно разместить в 8 битах байта). Для всех режимов по вертикали имеется 200 точек. Низкое разрешение (используемое только на PCjr) выводит 160 точек по горизонтали, среднее разрешение - вдвое больше (320 точек), а высокое разрешение - еще вдвое больше (640 точек). Число килобайт памяти, требуемое для каждого режима, приведено в [4.5.3].

В двух- и четырехцветном режимах PCjr имеет возможность выбрать любой из 16 доступных цветов. Цветной адаптер более ограничен. В двухцветном режиме он всегда ограничен белым и черным, а в четырехцветном режиме только цвет фона может выбираться из 16 цветов, в то время как основной цвет нужно брать только из двух предопределенных палитр. Палитра 0 содержит коричневый, зеленый и красный цвета, а палитра 1 - циан, магента и белый.

В отличие от текстовых данных в режимах 4-6 и 8-A графические данные разбиты на видеостранице на части. В большинстве режимов данные разбиваются на две части, при этом первая половина буфера содержит данные для четных строк экрана, а вторая - для нечетных (строки нумеруются начиная с верха экрана вниз). Однако в 16-цветных режимах PCjr буфер размером 32K делится на четыре части, каждая из которых содержит данные для каждой четвертой строки.

В 4-цветных режимах первый байт буфера содержит информацию о самых левых точках строки 0, причем старший бит отно-

сится к самой левой точке. Второй байт содержит информацию о следующем сегменте строки и т.д. Для всей строки требуется 80 байт. 81-й байт содержит информацию о левом конце строки 2. В 16-цветных режимах картина приблизительно такая же, но для каждой строки требуется 160 байт и каждая часть буфера содержит данные только для вдвое меньшего числа строк. Для цветного графического адаптера четные строки занимают память со смещениями от 0000 до 1F3FH, а нечетные - от 2000H до 3F3FH. Промежуток между 1F3FH и 2000H игнорируется. Для PCjr соответствующие ячейки могут существенно различаться в зависимости от режима и числа используемых страниц. PCjr специально устроен таким образом, что вывод в 16K, начинающихся с сегмента B800H, перенаправляется в ту область памяти, где реально расположен видеобуфер. Это свойство позволяет писать программы, которые будут одинаково работать на цветном дисплее и PCjr.

Для режимов экрана EGA от DH до 10H память организована совсем по-другому. Она разделяется на одну, две или четыре битовые плоскости, каждая из которых организована так же, как для черно-белого режима высокого разрешения, описанного выше: когда байт данных посылается в определенный адрес видеобуфера, каждый бит соответствует точке на экране, причем они описывают горизонтальный сегмент строки и бит 7 соответствует самой левой точке. Записываются четыре такие битовые плоскости, соответствующие одним и тем же адресам, в видеобуфере. Это отводит каждой точке 4 бита, что позволяет описывать 16 цветов. На рис. 4.4 показаны различные схемы распределения памяти.

В графическом режиме могут выводиться и символы. Однако они создаются не обычным способом. BIOS вырисовывает их поточечно, не изменяя фонового цвета. По этой причине такие вещи, как негативное изображение и мигание символов, недоступны в графическом режиме. Не выводится и курсор. BIOS может читать и определять установку точек в позиции курсора, чтобы узнать, какой символ там содержится. Символы располагаются в одной из позиций, соответствующих обычным строкам и столбцам. Это означает, что они всегда начинаются на границе, кратной восьми точкам.

4.4.1. Установка цветов для точечной графики

PCjr и EGA работают с цветом совсем по-другому, чем цветной адаптер. Они используют регистры палитры, которые позволяют в любой момент изменить цвет, соответствующий данному коду.

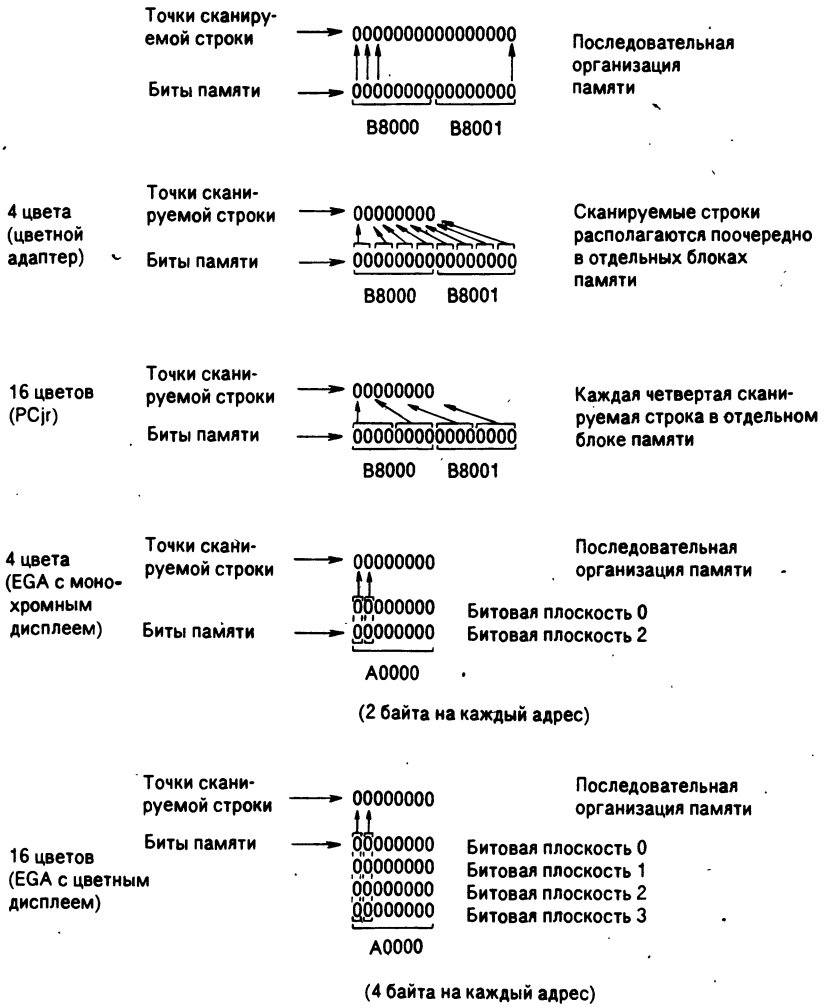


Рис. 4.4. Четыре способа представления графической информации в памяти

Вследствие этой разницы мы будем обсуждать обе системы отдельно и начнем с цветного адаптера.

Обе системы используют один и тот же основной набор кодов цвета, который в точности совпадает с применяемым в текстовых режимах:

Номер кода	Цепочка битов	Цвет
0	0000	черный
1	0001	синий
2	0010	зеленый
3	0011	циан
4	0100	красный
5	0101	магента
6	0110	коричневый
7	0111	белый
8	1000	серый
9	1001	ярко-синий
10	1010	ярко-зеленый
11	1011	яркий циан
12	1100	розовый
13	1101	яркая магента
14	1110	желтый
15	1111	ярко-белый

Для цветного графического адаптера цвет доступен только в режиме умеренного разрешения. Для каждой точки отводятся два бита каждого байта видеобуфера. Четыре возможные комбинации этих битов представляют один фоновый и три основных цвета. Фоновый цвет может быть любым из 16. Однако три основных цвета могут выбираться из одной из двух палитр, каждая из которых содержит только три предопределенных цвета. Это следующие цвета:

Номер кода	Цепочка битов	Палитра 0	Палитра 1
0	00	цвет фона	цвет фона
1	01	зеленый	циан
2	10	красный	магента
3	11	желтый/коричневый	белый

Если вы в какой-то момент переключились между палитрами, то все выведенные на экран цвета будут соответственно изменены.

Единственный способ использовать цвет, не входящий в эти палитры, состоит в том, чтобы искусственно рассматривать один из цветов палитры как фоновый цвет, что предполагает заполнение этим цветом всего экрана, когда экран чистится (используйте для этого прямое отображение в память). Истинный фоновый цвет может просматриваться "сквозь него" в качестве основного цвета. Такая техника приводит к созданию границы экрана, аналогичной той, что изображается в текстовых режимах. В противном случае граница экрана не может быть выделена цветом, так как весь экран закрашивается фоновым цветом, хотя точки, относящиеся к области границы, нельзя адресовать. Отметим, что BIOS хранит в своей области данных однобайтовую переменную, которая содержит текущий номер палитры. Ее адрес - 0040:0066H. Изменение этого числа не меняет текущую установку палитры; наоборот, если вы измените цвет палитры другими средствами, помимо функций операционной системы, то значение этой переменной будет модифицировано.

Символы могут перемешиваться с точечной графикой. Цвет, которым будут выводиться символы, зависит от того, какую функцию вы предпочтете для их вывода. Простейшая функция по умолчанию использует третий цвет текущей палитры. Однако имеется ряд способов выбрать любой из цветов палитры, а также выводить символы различными цветами. Эти проблемы обсуждаются в [4.1.3].

EGA и PCjr обеспечивают добавочную гибкость в работе с атрибутами цвета независимо от режима. При 16-цветной графике четыре бита, находящиеся в памяти для каждой точки экрана, дают цепочку битов, которая не переводится прямо в соответствующие цвета приведенной таблицы. Вместо этого каждый номер относится к одному из 16 регистров палитры. Каждый из этих регистров содержит цепочку битов, соответствующую цвету, который будет выводиться на самом деле. Если все 16 регистров будут содержать 0100, то независимо от того, какой атрибут приписан точке в памяти, она будет выведена красным цветом. Значение в регистре 0 используется в качестве фонового цвета. На рис. 4.1 в [4.1.3] показан этот механизм. В двух- и четырехцветном режимах необходимы только первые два или четыре регистра палитры.

Регистры палитры позволяют программе изменить цвет выводимого без каких-либо изменений в видеобуфере. Более того, отдельные объекты могут появляться и исчезать, как по волшебству. Это достигается изменением значения, содержащегося в регистре палитры, соответствующем данному объекту, на значение фонового цвета. Предположим, что фоновый цвет черный (0000) и что объект выведен с атрибутом 1110, так что он выводится в том

цвете, который указан в регистре палитры 15 (по умолчанию значение для этого регистра - желтый). Если изменить значение регистра 15 на 0000 (черный фоновый цвет), то объект исчезнет. Но на самом деле объект хранится в памяти, так как он записан с атрибутом 1110, а не с атрибутом 0000, как все точки фона. Объект может быть сделан снова видимым, если изменить значение регистра палитры 15 на 1110. Не обязательно, чтобы исчезали все желтые объекты, поскольку некоторые могут быть выведены с другим атрибутом, который также соответствует регистру палитры, содержащему желтый цвет.

EGA может использовать 6 бит регистра палитры, а не 4, когда к нему присоединен улучшенный цветной графический дисплей фирмы IBM. При этом становятся доступными 64 цвета, кодировка для которых R'G'B'RGB. R, G и B соответствуют темным цветам, а R', G' и B' - светлым. Различные комбинации создают 64 оттенка. Как всегда, 111111 соответствует белому цвету, а 000000 - черному. Отметим, что через регистры палитры для EGA всегда доступны 64 цвета независимо от того, в каком режиме он работает. При работе в режиме 4-цветной графики (как у цветного адаптера) активны только младшие 4 регистра палитры, но они могут содержать любые цвета.

Высокий уровень

Когда цветной дисплей работает в графическом режиме, Бейсик обрабатывает оператор COLOR по-другому, чем в текстовом режиме. Сначала идет фоновый цвет в виде числа от 0 до 15, а затем номер палитры 0 или 1. Например, COLOR 2,1 устанавливает зеленый фоновый цвет (#2) для всего экрана и активизирует палитру 1. После этого три возможных основных цвета указываются их номерами в палитре: 1 - циан, 2 - магента и 3 - белый (сравните с оператором PAINT). Чтобы выключить цвет в режиме умеренного разрешения, напишите SCREEN ,1. Отметим, что использование только черного и белого цветов в режиме умеренного разрешения не приводит к экономии памяти. PCjr использует оператор COLOR таким образом только в режиме SCREEN 1. Для режимов от SCREEN 3 до SCREEN 6 формат этого оператора COLOR основной,фоновый. При этом основной цвет соответствует числу в диапазоне от 1 до 15 в 16-цветном режиме и от 1 до 3 - в 4-цветном. Код цвета не должен быть равным 0, это значение зарезервировано для фонового цвета.

Существуют специальные операторы для установки регистров палитры: PALETTE и PALETTE USING. PALETTE устанавливает цвет, соответствующий любому атрибуту. Например, PALETTE 9,11 приводит к тому, что точки, нарисованные с цветом палитры

9 (обычно светло-синий), будут выведены в цвете 11 (светлый циан). Чтобы изменить установку всех регистров палитры к их первоначальному значению, т.е. чтобы регистр 0 содержал 0, регистр 12 - 12 и т.д., надо написать просто PALETTE. Отметим, что в режимах SCREEN 4 и SCREEN 6 регистры палитры инициализируются таким образом, чтобы атрибуты цветов 1-3 были такими же, как для палитры 1 на цветном графическом дисплее. Это делается в целях совместимости.

Все 16 регистров палитры могут быть установлены одним оператором PALETTE USING. PALETTE USING направляет содержимое 16-элементного целого массива в регистры палитры. Имея несколько таких массивов, программа может быстро переключать различные схемы цветов. Каждый элемент массива должен быть числом в диапазоне от 0 до 15 или -1. В последнем случае соответствующий регистр не изменяется. Например, для обращения привычной схемы цветов создайте массив, в котором ARRAYNAME(0) = 15, ARRAYNAME(1) = 14 и т.д. Затем напишите PALETTE USING ARRAYNAME(0) и содержимое массива ARRAYNAME будет передано в регистры палитры. 0 индицирует начальную позицию в массиве, с которой надо брать данные, посылаемые в регистры. Могут использоваться более длинные массивы, из которых данные можно брать начиная с любой точки при условии, что до конца массива еще есть 16 элементов. PALETTE USING ARRAYNAME(12) будет брать данные начиная с 12-го байта массива. Отметим, что оператор PALETTE USING работает как в текстовом, так и в графическом режимах. Вот пример:

```
100 DEF INT A-Z                         'все переменные целые
110 DIM SCHEME1(16)                     'массив для схемы цветов #1
120 DIM SCHEME2(16)                     'массив для схемы цветов #2
130 DATA 3,5,9,2,4,12,15,1,6,7,14,13,8,11,10,0
140 DATA 0,11,13,7,1,12,2,5,10,8,14,6,15,4,9,3
150 FOR N = 0 TO 15                     'для каждого регистра палитры
160 READ Q                              'прочитать код цвета
170 SCHEME1(N) = Q                      'и поместить его в массив
180 NEXT
190 FOR N = 0 TO 15                     'то же самое со вторым массивом
200 READ Q
210 SCHEME2(N) = Q
220 NEXT
230 PALETTE USING SCHEME1(0)            'установка регистров

500 PALETTE USING SCHEME2(0)            'меняем их посреди программы
```

Средний уровень

Функция BH прерывания 10H устанавливает как фоновый цвет, так и цвета палитры, но не одновременно. Для установки фонового цвета надо поместить в BH 0, а затем код цвета от 0 до 15 - в BL. Для установки палитры надо поместить в BH 1, а в BL 0 или 1. В данном примере устанавливается цвет фона циан и выбирается палитра 0:

```
;---установка цвета фона и палитры
MOV  AH,0BH          ;функция установки цвета
MOV  BH,0            ;сначала устанавливаем фоновый цвет
MOV  BL,3            ;код циана
INT  10H             ;установка цвета
MOV  BH,1            ;теперь устанавливаем палитру
MOV  BL,1            ;выбираем палитру 1
INT  10H             ;устанавливаем палитру
```

На PCjr эта функция работает точно так же в 4-цветном режиме, устанавливая регистры 1-3 в одну из схем цветов, используемых цветным адаптером. В 2-цветном режиме 0 в BL соответствует белому цвету, как цвету 1, а 1 - черному. Эта функция не влияет на назначения, используемые в 16-цветном режиме. Однако во всех случаях фоновый цвет может быть установлен засылкой в BH 0, а в BL - кода цвета.

Низкий уровень

Для цветного адаптера мы можем получить доступ к "регистру выбора цвета" через порт 3D9H. В графических режимах этот регистр действует по-другому, чем в текстовых (описанных в [4.1.3]). Биты 0-3 содержат информацию о фоновом цвете в обычном формате (соответственно синий, зеленый и красный компоненты и интенсивность). Бит 5 выбирает палитру, и когда этот бит равен 0, то назначается палитра номер 0. В графических режимах остальные биты не имеют значения. Этот регистр только для записи, поэтому вы должны указывать информацию и о фоновом цвете, и о палитре при изменении любого из них.

```
MOV  DX,3D9H         ;адрес регистра выбора цвета
MOV  AL,00100110B    ;цепочка битов для циана и палитры 1
OUT  DX,AL           ;посылаем ее
```

Поскольку PCjr и EGA используют регистры палитры, данный пример к ним неприменим. Для них надо просто загрузить требу-

емые значения в эти регистры. У PCjr такие регистры нумеруются от 10H до 1FH. Доступ ко всем регистрам осуществляется через один порт с адресом 3DAH. Любое новое значение, принимаемое этим портом, воспринимается адресным регистром. Поэтому надо послать сначала номер регистра, а затем код цвета. Чтобы быть уверенным, что порт ожидает номер регистра, надо прочитать из него. В следующем примере ярко-синий цвет (1001) помещается в регистр палитры 2:

```
;---помещаем код ярко-синего цвета в регистр палитры 2
MOV  DX,3DAH          ;адрес массива ворот дисплея
IN   AL,DX            ;читаем из него
MOV  AL,12H           ;номер регистра
OUT  DX,AL            ;посылаем номер регистра
MOV  AL,00001001B     ;код ярко-синего цвета
OUT  DX,AL            ;посылаем цвет
```

У EGA адрес порта доступа к регистрам палитры - 3C0H, а регистры нумеруются от 00 до 0FH. Надо прочитать из порта 3DAH (а не 3C0H), чтобы быть уверенным, что ожидается номер регистра. Когда к EGA присоединен улучшенный цветной дисплей и переключатели установлены соответствующим образом, в регистры помещаются 6-битовые значения.

4.4.2. Рисование точки на экране (монохромный, цветной адаптеры и PCjr)

Вследствие организации графической информации в видеобуфере вывод одной точки подразумевает изменение отдельных битов памяти. Режимы двух, четырех и шестнадцати цветов требуют, чтобы для установки характеристик одной точки были изменены один, два и четыре бита соответственно. Для этих операций необходимы огромные затраты процессорного времени, поэтому графическое программное обеспечение, как правило, работает очень медленно. Тщательное обдумывание часто позволяет сразу установить все биты одного байта, а не обращаться к одному и тому же байту 4 или 8 раз. Имейте это в виду и не следуйте слепо приведенной здесь технике поточечного вывода.

Высокий уровень

Бейсик предоставляет операторы PSET и PRESET для изменения цвета отдельной точки. Эти имена образованы от PointSET (установка точки) и PointRESET (сброс точки). Они очень

похожи. За ними должны следовать координаты столбца и строки, указываемой точки, заключенные в скобки. Отметим, что координаты следуют в порядке x,y - т.е. сначала идет столбец, а затем строка; этот порядок обратный по отношению к порядку оператора LOCATE, который позиционирует текст на экране. PSET(50,80) или PRESET(50,80) устанавливают цвет точки в столбце 50 и строке 80. За оператором PSET может следовать код цвета, который лежит в диапазоне, определяемом текущим режимом экрана. Если код цвета не указан, то берется максимальный номер кода, который допустим для данного режима. В PRESET цвет не указывается. Он всегда возвращает точке цвет фона (код 0). Например:

```
100 PSET(100,180),3          'установка цвета 3 текущей палитры
110 PRESET(100,180)          'изменение цвета точки на фоновый
```

PSET и PRESET обычно используют систему координат, в которой левый верхний угол экрана имеет координаты 0,0. Оператор WINDOW позволяет переопределить систему координат так, что, например, координаты левого верхнего угла будут 100,100, центра экрана 0,0, а правого нижнего угла - 100,-100. Для этого случая надо записать оператор в виде WINDOW(-100,100)-(100,-100). (Новые координаты не будут влиять на систему координат 25x80 (или 25x40), в которой оператор LOCATE позиционирует символы на графическом экране [4.2.1].)

Как и в операторе LINE [4.4.5], первое число каждой пары в скобках указывает горизонтальную координату (по оси x). Координаты могут быть как положительными, так и отрицательными, лишь бы они не были равными. Левому краю экрана всегда присваивается меньшее число (которое может быть большим отрицательным). Таким образом, даже если вы поменяете координаты в примере и запишете оператор WINDOW(100,-100)-(-100,100), то значение -100 будет взято для левой границы экрана.

Второе число каждой пары координат определяет границы экрана по вертикали. И опять меньшее значение будет относиться к нижней границе экрана независимо от того, в какой паре координат оно указано. Большее положительное значение (или меньшее из двух отрицательных) присваивается в качестве значения оси y для верхней строки экрана. Направление увеличения значений может быть обращено, с тем чтобы максимальные значения соответствовали низу экрана и наоборот. Надо просто добавить к оператору слово SCREEN, например WINDOW SCREEN(-100,100)-(100,-100).

Программа может указывать точки, относящиеся к области за пределами координат экрана. Например, центр окружности может находиться за пределами экрана, с тем чтобы видна была только

часть дуги. Отметим, что координаты, указываемые оператором WINDOW, могут непрерывно изменяться при изменении масштаба или угла зрения на объект. Изображение должно перерисовываться, а иногда стираться, при изменении координат окна.

Оператор PMAP преобразует координаты от обычной физической системы координат к "мировой" системе, устанавливаемой оператором WINDOW. PMAP использует четыре кодовых номера:

0	преобразует x из "мировой" системы в физическую
1	преобразует y из "мировой" системы в физическую
2	преобразует x из физической системы в "мировую"
3	преобразует y из физической системы в "мировую"

Оператор имеет форму PMAP(позиция,код). Предположим, что вы установили систему "мировых" координат оператором WINDOW. Координаты левого верхнего угла экрана - (-100,100), а правого нижнего - (100,-100). Какой будет позиция центральной точки экрана (0,0) при использовании обычной физической системы 320x200, в которой левый верхний угол имеет координаты 0,0? Чтобы найти X, запишите X = PMAP(0,0), а Y - запишите Y = PMAP(0,1). Вы получите значение X = 160, а Y = 100.

Средний уровень

Функция CH прерывания 10H устанавливает точку. DX содержит строку, а CX - столбец. Они отсчитываются от 0. Код цвета помещается в AL. Отметим, что содержимое AX будет разрушено при выполнении прерывания. Если вы используете это прерывание в цикле, то не забудьте сохранить AX на стеке и каждый раз восстанавливать его.

```
;---вывод точки с координатами 100,180
MOV  AH,0CH          ;функция установки точки
MOV  AL,3            ;выбираем цвет 3 палитры
MOV  CX,100          ;строка
MOV  DX,180          ;столбец
INT  10H             ;выводим точку
;---стираем точку
MOV  AH,0CH          ;восстанавливаем функцию
MOV  AL,0            ;используем для стирания фоновый цвет
MOV  DX,100          ;строка
```

```
MOV  CX,180          ;столбец
INT  10H             ;стираем точку
```

В то время как цвет палитры помещается в младшие биты AL, старший бит также имеет значение. Если он равен 1, то над цветом производится операция исключающего ИЛИ с текущим цветом. Напомним, что операция исключающего ИЛИ устанавливает бит только в том случае, если из двух сравниваемых битов установлен только один. Если оба сравниваемые бита равны 1 или оба равны 0, то результат будет 0. Для двухцветного режима это означает, что такая операция обращает установку бита. Если эту операцию применить ко всем точкам экрана, то будет обращен весь экран. В 4- и 16-цветном режимах, с другой стороны, области экрана могут менять свои цвета. Например, пусть в 4-цветном режиме умеренного разрешения область занята точками либо цвета 1 палитры (установка битов 01B) или цвета 2 палитры (10B). Что произойдет, если применить ко всем точкам этой области операцию исключающего ИЛИ с 11B? 01B перейдет в 10B, а 10B перейдет в 01B - цвета будут обращены.

Низкий уровень

На низком уровне мы имеем возможность прямого доступа к видеобуферу (отображение в память). Сначала вы должны вычислить смещение точки (а) внутри буфера и (б) внутри байта, содержащего биты, относящиеся к данной точке. После этого битовые операции обеспечат соответствующую установку. Отметим, что если вы станете применять эту технику на PCjr, когда он работает в одном из 16-цветных режимов, использующих страницу размером 32K, то вывод в адреса, начинающиеся с параграфа B800H, не будет перенаправлен верно. Вам необходимо прямо адресовать реальные ячейки, расположенные в сегменте ниже 2000H.

Для нахождения точки нужно прежде всего определить, находится ли она в четной или нечетной строке. В данном примере строка помещена в CX, а столбец - в DX. Если бит 0 регистра CX равен 0, то строка имеет четный номер. Четные строки расположены со смещением 0 относительно начала буфера. Если же строка имеет нечетный номер, то следует добавить смещение 2000H для указания на начало второй половины буфера.

Затем разделите номер строки на 2, подсчитайте число только четных или нечетных строк и умножьте результат на 80, так как на одну строку расходуется 80 байт. Для деления можно использовать инструкцию SHL, а результат даст общее число байтов во

всех строках, предшествующих строке, в которой расположена искомая точка.

Вместо того чтобы вычислять число столбцов в текущей строке, лучше сначала определить позицию пары битов в байте, которые содержат эту точку. Это достигается обращением всех битов в номере столбца (после того, как сохранена его копия) и выделением двух младших битов. Эта процедура покажет, находятся ли два бита, относящиеся к точке на первой, второй, третьей или четвертой позиции, в байте. Умножив это значение на 2, мы получаем номер в байте первого из двух битов, относящихся к данной точке.

Затем приходит время подсчитать число байтов в строке, предшествующих байту, содержащему информацию о требуемой точке. Для режима умеренного разрешения надо разделить число столбцов на 4, а для высокого разрешения - на 8. После этого надо сложить три смещения: смещение за счет номера строки, за счет номера столбца и смещение начала четных/нечетных строк в буфере. После этого вы можете получить требуемый байт из буфера.

Наконец, надо произвести операцию над соответствующими битами байта. Вращайте байт до тех пор, пока пара битов, относящихся к точке, не станет младшей. При вращении необходимо использовать ранее подсчитанное значение позиции битов. Затем выключите оба бита, поместив в них с помощью инструкции OR требуемый код палитры. Затем произведите обратное вращение и пошлите байт обратно в буфер.

```
;---в сегменте данных
PAL_COL DB   2
;---вызов процедуры
        MOV AX,0B800H       ;указываем на видеобуфер
        MOV ES,AX
        MOV CX,100          ;номер строки
        MOV DX,180          ;номер столбца
        CALLSET_DOT

.

.

;---определяем число байтов в предшествующих строках
SET_DOT PROC
        TEST CL,1           ;номер строки нечетный?
        JZ   EV_ROW         ;если нет, то вперед
        MOV BX,2000H        ;смещение для нечетных строк
        JMP  SHORT CONT     ;переход вперед
```

```
EV_ROW: MOV BX,0            ;смещение для четных строк
CONT:   SHR CX,1            ;делим число строк на 2
        MOV AL,80           ;умножаем на 80
        MUL CL              ;в AX - число байтов
;---определяем положение пары бит в байте
        MOV CX,DX           ;копируем номер столбца
        NOT CL              ;обращаем биты
        AND CL,00000011B    ;в CL - позиция битов (0-3)
        SHL CL,1            ;позиция первого бита пары
;---подсчитываем смещение столбца в байтах
        SHR DX,1            ;делим номер столбца на 4
        SHR DX,1            ;(нужны два младших бита)
;---вычисляем смещение для изменяемого байта
        ADD AX,DX           ;складываем все три смещения
        ADD BX,AX
;---изменяем биты нужного байта
        MOV AH,ES:[BX]      ;читаем нужный байт
        ROR AH,CL           ;сдвигаем нужные биты вниз
        AND AH,11111100B    ;чистим младшие 2 бита
        MOV AL,PAL_COL      ;изменяем их на цвет палитры
        OR  AH,AL
        ROL AH,CL           ;обратное вращение
        MOV ES:[BX],AH      ;возвращаем байт
        RET
SET_DOT ENDP
```

4.4.3. Рисование точки на экране (EGA)

У EGA-адаптера графика более сложная. С точки зрения процессора режимы экрана 0-7 действуют так же, как соответствующие режимы для цветного адаптера или PCjr, но режимы от DH до 10H совершенно другие. Организация памяти для этих режимов меняется в зависимости от числа цветов и количества памяти, имеющейся на плате дисплея (см. рис. 4.4.)

В режимах D, E и 10H память разбита на 4 битовые плоскости. Каждая плоскость организована таким же образом, как для черно-белого режима высокого разрешения цветного адаптера, который обсуждался в [4.4.2]: когда байт данных посылается в определенный адрес видеобуфера, каждый бит соответствует точке на экране, причем весь байт соответствует горизонтальному сегменту линии, а бит 7 - самой левой точке. Выводятся четыре такие бито-

вые плоскости, относящиеся к одним и тем же адресам в видеобуфере. Это приводит к тому, что каждая точка описывается четырьмя битами (давая 16 цветов), причем каждый бит находится в отдельном байте отдельной битовой плоскости.

Но как вы можете записать 4 различных байта данных, расположенных по одному и тому же адресу? Ответ на этот вопрос состоит в том, что вы не посылаете последовательно четыре байта по этому адресу. Вместо этого один из трех режимов записи позволяет изменить все 4 байта на основании одного байта данных, полученного от процессора. Влияние данных, посланных процессором, зависит от установки нескольких регистров, включающих два регистра маски, которые определяют, на какие биты и в каких битовых плоскостях будут изменяться биты.

Для понимания этих регистров мы должны сначала разобраться с четырьмя регистрами задвижки (latch register). Они содержат данные для четырех битовых плоскостей в той позиции, к которой было последнее обращение. (Заметим, что термин "битовая плоскость" используется как для целой области видеобуфера, так и для однобайтового буфера, временно хранящегося в регистре задвижки.) Когда процессор посылает данные по определенному адресу, эти данные могут изменить или полностью сменить данные регистра задвижки, а впоследствии именно данные из регистра задвижки записываются в видеобуфер. Каким образом данные процессора влияют на регистр задвижки, зависит от режима записи, а также от установки некоторых других регистров. При чтении адреса из видеобуфера регистры задвижки заполняются четырьмя байтами из четырех битовых плоскостей по данному адресу. Регистрами задвижки легко манипулировать, производя с их содержимым различные логические операции, что позволяет устраивать различные графические трюки.

Регистр маски битов и регистр маски карты действуют на регистры задвижки, защищая определенные биты или битовые плоскости от изменения под действием данных, поступающих от процессора. Регистр маски битов - это регистр только для записи, адрес порта которого 3CFH. Сначала надо послать 8 в порт 3CEH, чтобы указать на этот регистр. Установка бита этого регистра в 1 маскирует этот бит во всех четырех битовых плоскостях, делая соответствующую точку недоступной для изменения. Однако поскольку оборудование работает в байтовых терминах, реально "неизменяемые" биты перезаписываются в четыре битовые плоскости. Данные для этих маскируемых битов хранятся в регистрах задвижки, поэтому программа должна быть уверена, что текущее содержимое регистров задвижки относится к правильному адресу памяти. По этой причине перед записью по данному адресу надо считывать из него.

Регистр маски карты имеет адрес порта 3C5H. Этот регистр только для записи. Перед посылкой данных надо послать по этому адресу 2 как указатель. Биты 0-3 этого регистра соответствуют битовым плоскостям 0-3; старшие 4 бита регистра не используются. Когда биты 0-3 равны 0, соответствующие битовые плоскости не изменяются при операциях записи. Это свойство используется по-разному в различных режимах записи, как вы увидите в дальнейшем.

Три режима записи устанавливаются регистром режима, который является регистром только для записи и который индексируется предварительной засылкой *5* в порт с адресом 3CFH. Режим записи устанавливается в битах 0 и 1, как число от 0 до 2. Бит 2 должен быть равным 0, так же как и биты 4-7. Бит 3 устанавливает один из двух режимов чтения из видеобуфера. Этот бит может быть 0 или 1. BIOS EGA устанавливает режим записи в 00.

Режим записи 0

В простейшем случае режим записи 0 копирует данные процессора в каждую из четырех битовых плоскостей. Пусть, например, по определенному адресу видеобуфера послано 11111111B и разрешены все биты и все битовые плоскости (т.е. ничто не маскировано описанными выше регистрами масок). Тогда каждый бит во всех четырех плоскостях будет установлен в 1, так что цепочка битов для каждой из соответствующих точек будет 1111B. Это означает, что 8 точек будут выведены в цвете 15, который изначально соответствует ярко-белому цвету, хотя регистры палитры позволяют, чтобы на самом деле это был любой из допустимых цветов.

Теперь рассмотрим ту же ситуацию, когда посылается значение 00001000B. Цепочка битов для точки 3 будет 1111, а для остальных - 0000, что соответствует черному (изначально). Поэтому в данном случае только точка 3 появится на экране (ярко-белая), а остальные 7 точек будут выключены. Даже если остальные 7 точек перед этим выводились в каком-то цвете, теперь все они будут переключены на 0000.

Рассмотрим другие цвета, кроме 1111B. Если вы пошлете код палитры желаемого цвета в регистр маски карты, то регистр будет маскировать определенные битовые плоскости таким образом, что будет воспроизведен требуемый цвет. Например, если вы хотите цвет с кодом 0100, то пошлите 0100 в регистр маски карты. Тогда битовые плоскости 0, 1 и 3 не будут изменяться. Когда вы пошлете по нужному адресу 11111111B, это значение будет помещено только в битовую плоскость 2, и цепочка битов для каждой точки будет 0100. Если вы пошлете по этому адресу

00001000B, то точка 3 будет иметь цепочку битов 0100, а остальные точки - 0000.

Существует, однако, одна сложность. Регистр маски карты запрещает изменение битовых плоскостей, но не обнуляет их. Предположим, что битовая плоскость 0 была заполнена единицами, а битовые плоскости 1 и 3 были заполнены нулями. Если вы запретите изменения в этих трех плоскостях, а затем пошлете 11111111B по определенному адресу, то битовая плоскость 2 будет заполнена 11111111B, а битовая плоскость 0 сохранит свои единицы, поэтому результирующий код цвета каждой точки станет 0101B. Встречаются случаи, когда это свойство можно использовать для изменения цветов экрана. Но, вообще говоря, необходимо очищать все четыре битовые плоскости (т.е. все четыре регистра задвижки) перед тем, как писать туда любые цвета, кроме 1111B или 0000B. Это делается просто посылкой 0 по указанному адресу. Необходимо, чтобы при этом была разрешена запись во все четыре битовые плоскости.

Приведенное выше обсуждение касалось одновременного вывода восьми точек. Ну а как вывести меньшее количество точек? В этом случае, конечно, необходимо сохранить существующие данные для некоторых точек, а чтобы это было возможно, текущее содержимое данного адреса сохраняется в регистрах задвижки. Затем используется регистр маски битов для маскирования тех точек, которые не должны изменяться. Если бит этого регистра сброшен в 0, то данные, получаемые от процессора для этого бита, игнорируются и вместо них используются данные, хранящиеся в регистрах задвижки. Равен ли этот бит в данных процессора 0 или 1 - не имеет значения; если вы изменяете только бит 2, а все остальные маскированы, то данные, которые приходят от процессора, могут быть 0FFH или 4H, или любое другое значение, для которого бит 2 установлен. Если бит 2 сброшен, то 0 помещается в этой позиции во всех разрешенных битовых плоскостях.

Вообще говоря, программа должна сначала прочитать любую ячейку, в которую она собирается записать меньше 8 точек. Имеются два режима чтения (обсуждаемые в [4.4.4]) и безразлично, какой из них выбран. Операция чтения загружает регистры задвижки четырьмя байтами данных для нужного адреса памяти. Данные, возвращаемые процессору операцией чтения, могут быть отброшены.

До сих пор были рассмотрены самые простые возможности режима записи 0. При желании вы можете делать намного более сложные манипуляции. Одна из возможностей состоит в модификации регистров задвижки с помощью логических операций перед

записью. Для реализации этой возможности регистр вращения данных использует следующие биты:

биты	2-0		число вращений
	4-3	00	данные не модифицируются
		01	логическое И с регистром задвижки
		10	логическое ИЛИ с регистром задвижки
		11	исключающее ИЛИ с регистром задвижки
	7-5		не используются

Число вращений, которое может быть от 0 до 7, показывает, сколько битов данных должны вращаться перед тем, как поместить их в регистр задвижки. Обычно это значение равно нулю. Аналогично биты 4-3, как правило, равны 00, кроме случаев, когда производятся логические операции. За счет манипуляций с этим регистром одни и те же данные могут выдавать различные цвета и изображения без дополнительной процессорной обработки. Регистр вращения данных индексируется посылкой 3 в порт 3CEH; затем данные посылаются в 3CFH.

Наконец, режим записи 0 может работать совсем по-другому, если разрешены установка/сброс. В этом случае определенные цвета в младших четырех битах регистра установки/сброса (который тоже имеет адрес порта 3CFH, а индексируется посылкой 0 в 3CEH). Имеется соответствующий регистр разрешения установки/сброса, который разрешает любой из этих четырех битов, устанавливая свои младшие биты в 1. Когда все 4 бита в регистре установки/сброса разрешены, они помещаются во все 8 адресов битовой плоскости при получении данных от процессора, при этом сами данные процессора отбрасываются. Если разрешены не все биты установки/сброса, то данные процессора помещаются для запрещенных точек. Отметим, что регистр маски битов запрещает запись данных установки/сброса в определенные точки, но установка регистра маски карты игнорируется при использовании установки/сброса. BIOS инициализирует регистр разрешения установки/сброса в 0, так что он неактивен. Его адрес порта 3CFH, а индексируется он посылкой 1 в порт 3CEH.

Режим записи 1

Режим записи 1 предназначен для специальных приложений. В этом режиме текущее содержимое регистра задвижки записывается по указанному адресу. Напоминаем, что регистры задвижки заполняются операцией чтения. Этот режим очень полезен для быстрого

переноса данных при операциях сдвига экрана. Регистр маски битов и регистр маски карты не влияют на эту операцию. Не имеет значения также, какие данные посылает процессор - содержимое регистров задвижки записывается в память без изменений.

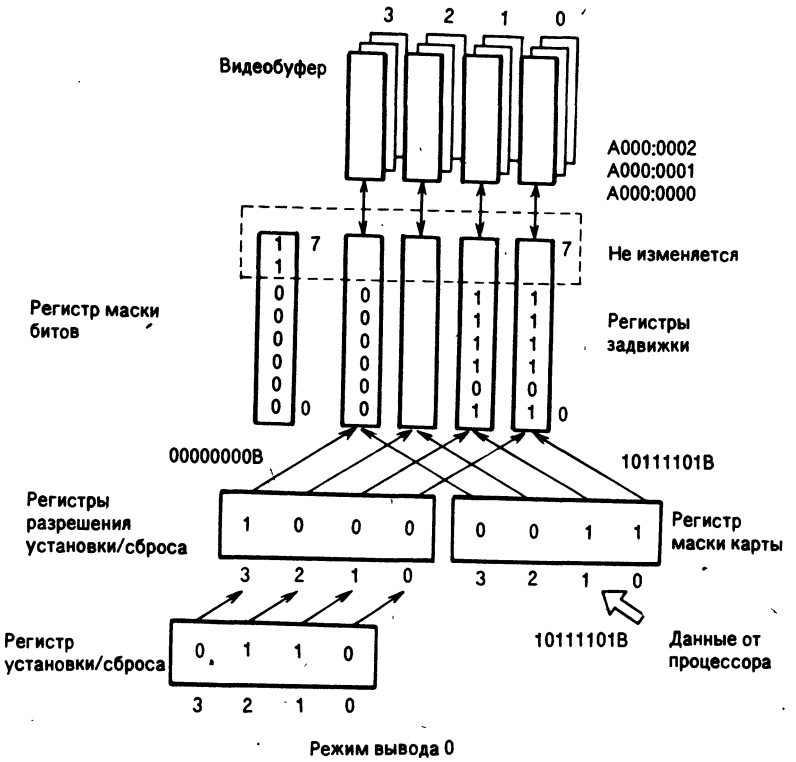


Рис. 4.5. Графические режимы вывода EGA

Режим записи 2

Режим записи 2 предоставляет альтернативный способ установки отдельных точек. Процессор посылает данные, у которых имеют значение только 4 младших бита, рассматриваемые как цвет (индекс регистра палитры). Можно сказать, что эта цепочка

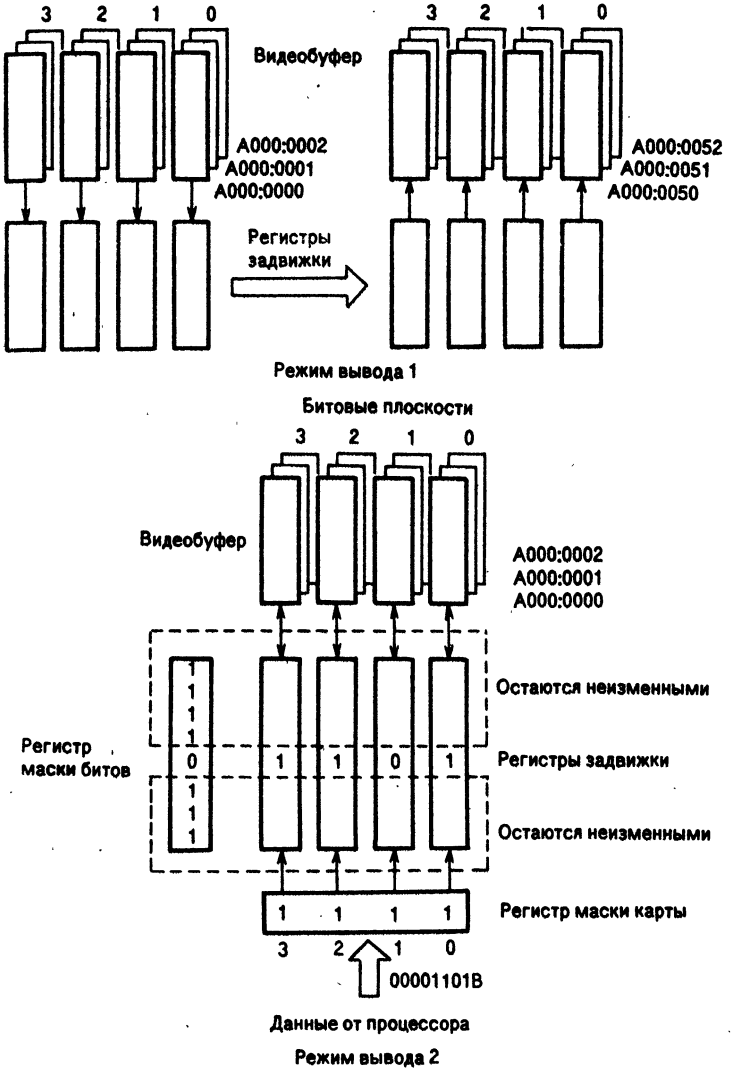


Рис. 4.5. Графические режимы вывода EGA (продолжение)

битов вставляется поперек битовых плоскостей. Цепочка дублируется на все восемь точек, относящихся к данному адресу, до тех пор, пока регистр маски битов не предохраняет определенные

точки от изменения. Регистр маски карты активен, как и в режиме записи 0. Конечно, процессор должен послать полный байт, но только младшие 4 бита существенны.

Высокий уровень

Бейсик поддерживает EGA в традиционных режимах цветного графического адаптера. Ко времени выхода этой книги поддержки дополнительных режимов EGA не существовало. Поэтому у вас нет другого выхода, кроме как использовать прямое отображение в видеобуфер, который начинается с адреса A000:0000. Самая тяжелая проблема состоит в установке режима дисплея. Для ее решения используйте следующую процедуру на машинном языке:

```
10 S$ = CHR$(&H2A) + CHR$(&HE4) + CHR$(&HB0) + CHR$(&H0D)
 + CHR$(&HCD) + CHR$(&H10) + CHR$(&HCB)
20 DEF SEG                          'установка сегмента
30 Y = VARPTR(S$)                   'указатель на строку
40 Z = PEEK(Y + 1) + PEEK(Y + 2)*256 'вычисление адреса строки
50 CALL Z                           'вызов процедуры
```

Четвертый байт S$ содержит номер режима, в данном случае режим D. Вы можете выбрать другой режим. В приложении Г объясняется, как эта процедура работает в Бейсике. Она полностью завершенная, не нужно никакой побочной памяти, в которой содержался бы машинный код. Не забудьте восстановить режим дисплея после завершения своих манипуляций.

Затем надо установить соответствующий режим записи. Вот как устанавливается режим записи 2:

```
50 OUT &H3CE,5          'индексируем регистр режима записи
60 OUT &H3CF,2          'выбираем режим 2
```

Режим записи также должен быть восстановлен после завершения программы.

Наконец, приведем образцы кода, реализующие прямое отображение в видеобуфер:

Режим записи 0:

```
100 'рисуем красную точку в левом верхнем углу экрана
110 DEF SEG = &HA000        'указываем на видеобуфер
120 OUT &H3CE,8             'адресуем регистр маски битов
```

```
130 OUT &H3CF,128          'маскируем все биты, кроме седьмого
140 X = PEEK(0)            'читаем текущее значение в задвижку
150 POKE 0,0               'чистим
160 OUT &H3C4,2            'адресуем регистр маски карты
170 OUT &H3C5,4            'устанавливаем красный цвет
180 POKE 0,&HFF            'рисуем точку
```

Режим записи 1:

```
100 'копируем верхнюю строчку точек в следующую
110 DEF SEG = &HA000       'указываем на видеобуфер
120 FOR N = 0 TO 79        'для всех 80 байт  строки
130 X = PEEK(N)            'заполняем задвижки
140 POKE N + 80,Y          'копируем в следующую строку
150 NEXT                   'переходим к следующему сегменту
```

Режим записи 2:

```
100 'рисуем красную точку в левом верхнем углу экрана
110 DEF SEG = &HA000       'указываем на видеобуфер
120 OUT &H3CE,8            'адресуем регистр маски битов
130 OUT &H3CF,128          'маскируем все биты, кроме седьмого
140 X = PEEK(0)            'читаем текущее значение в задвижку
150 POKE 0,4               'посылаем красный цвет
```

Средний уровень

EGA поддерживает стандартные графические функции BIOS. Можно вывести точку с помощью функции CH прерывания 10H, так же как для цветного дисплея или PCjr. При входе DX должен содержать номер строки, а CX - номер столбца, и то и другое отсчитывается от 0. Код цвета помещается в AL. Содержимое AX меняется при выполнении прерывания.

```
;---рисуем точку по адресу 50,100
MOV  AH,0CH          ;функция вывода точки
MOV  AL,12           ;выбираем регистр палитры 12
MOV  CX,100          ;номер строки
MOV  DX,50           ;номер столбца
INT  10H             ;рисуем точку
```

Низкий уровень

Ниже приведены примеры для трех режимов записи. Перед их выполнением необходимо установить режим дисплея, использующий видеобуфер с адреса A000:0000. Для этого можно применить стандартную функцию BIOS, например, для установки режима D:

```
MOV  AH,0          ;функция установки режима
MOV  AL,0DH        ;выбираем режим D
INT  10H           ;устанавливаем режим
```

Не забудьте восстановить режим перед завершением программы. Кроме того, необходимо установить требуемый режим записи. Вот пример установки режима записи 2:

```
MOV  DX,3CEH       ;указываем на регистр адреса
MOV  AL,5          ;индексируем регистр 5
OUT  DX,AL         ;посылаем индекс
INC  DX            ;указываем на регистр режима
MOV  AL,2          ;выбираем режим записи 2
OUT  DX,AL         ;устанавливаем режим
```

И наконец, примеры трех режимов записи:
Режим записи 0:

```
;---рисуем красную точку в левом верхнем углу экрана
MOV  AX,0A000H     ;указываем на видеобуфер
MOV  ES,AX
MOV  BX,0          ;указываем на первый байт буфера
;---маскируем все биты, кроме седьмого
MOV  DX,3CEH       ;указываем на адресный регистр
MOV  AL,8          ;номер регистра
OUT  DX,AL         ;посылаем его
INC  DX            ;указываем на регистр данных
MOV  AL,10000000B  ;маска
OUT  DX,AL         ;посылаем данные
;---чистим текущее содержимое задвижки
MOV  AL,ES:[BX]    ;читаем содержимое в задвижку
MOV  AL,0          ;готовимся к очистке
MOV  ES:[BX],AL    ;чистим задвижку
```

```
;---установка регистра маски карты для красного цвета
MOV  DX,3C4H          ;указываем на адресный регистр
MOV  AL,2             ;индекс регистра маски карты
OUT  DX,AL            ;установка адреса
INC  DX               ;указываем на регистр данных
MOV  AL,4             ;код цвета
OUT  DX,AL            ;посылаем код цвета
;---рисуем точку
MOV  AL,0FFH          ;любое значение с установленным 7 битом
MOV  ES:[BX],AL       ;выводим точку
```

Режим записи 1:

```
;---копируем строку в следующую строку
        MOV  CX,80            ;число байтов в строке
        MOV  BX,0             ;начинаем с 1-го байта буфера
        MOV  AX,0A000H        ;адрес буфера
        MOV  ES,AX
NEXT:   MOV  AL,ES:[BX]       ;заполняем задвижку
        MOV  ES:[BX] + 80,AL  ;выводим в следующую строку
        INC  BX               ;переходим к следующему байту
        LOOP NEXT ;
```

Режим записи 2:

```
;---рисуем красную точку в левом верхнем углу экрана
MOV  AX,0A000H        ;адрес буфера
MOV  ES,AX
MOV  BX,0             ;указываем на первый байт буфера
;---установка регистра маски битов
MOV  DX,3CEH          ;указываем на адресный регистр
MOV  AL,8             ;регистр маски битов
OUT  DX,AL            ;адресуем регистр
INC  DX               ;указываем на регистр данных
MOV  AL,10000000B     ;маскируем все биты, кроме 7-го
OUT  DX,AL            ;посылаем данные
;---рисуем красную точку
MOV  AL,ES:[BX]       ;заполняем регистры задвижки
MOV  AL,4             ;красный цвет
MOV  ES:[BX],AL       ;рисуем точку
```

4.4.4. Определение цвета точки экрана

Для графических режимов цветного адаптера или PCjr определение цвета точки на низком уровне состоит в обращении процедуры вывода точки: программа читает из видеобуфера и выделяет интересующие биты. Однако для EGA этот метод непригоден, поскольку в режимах DH - 10H каждому адресу памяти соответствует два или четыре байта. EGA имеет два режима чтения, чтобы преодолеть эту трудность. Имейте в виду, что для PCjr и EGA после того, как вы определили код цвета точки, необходимо еще проверить установку текущего регистра палитры для этого кода, чтобы определить, какой цвет ему приписан.

Любой язык программирования имеет доступ к двум режимам чтения EGA. В режиме 0 возвращается байт, содержащийся во всех четырех битовых плоскостях, по указанному адресу. Режим 1 ищет указанный код цвета и возвращает байт, в котором бит установлен в 1, когда соответствующая точка имеет данный цвет. Бит 3 регистра режима определяет, какой режим чтения установлен (0 = режим 0). Доступ к этому регистру осуществляется через порт 3CFH, и вы должны предварительно послать 5 в порт 3CEH, чтобы выбрать этот регистр. Обычно все остальные биты этого регистра, который можно только писать, сброшены в 0, кроме битов 0 и 1, которые определяют режим записи. Поскольку при инициализации BIOS устанавливает эти биты в режим записи 0 (так что они оба равны 0), вам нужно просто послать в этот регистр 0, чтобы установить режим чтения 0, и послать 8, чтобы установить режим чтения 1.

Режим чтения 0 требует, чтобы вы предварительно установили регистр выбора карты. Единственная задача этого регистра - установить, какая из карт битов должна быть прочитана. Поэтому в него надо послать число от 0 до 3. Этот регистр имеет адрес порта 3CFH и надо предварительно послать 4 в порт 3CEH, чтобы указать этот регистр.

Режим чтения 1 более сложен. Сначала регистр сравнения цветов должен быть заполнен цепочкой битов для кода цвета, который вы ищете. Этот код помещается в младшие 4 бита регистра; старшие 4 бита несущественны. Этот регистр имеет адрес порта 3CFH и указывается предварительной засылкой 2 в порт 3CEH. После чтения ячейки памяти возвращается байт, который имеет биты, установленные в 1 для каждой точки, имеющей нужный цвет. Однако за счет использования регистра безразличия цвета (color don't care register) один или более битов кода цвета могут при сравнении игнорироваться. Обычно 4 младших бита этого регистра установлены в 1; обнуление одного из этих битов приведет к тому, что содержимое соответствующей битовой плос-

кости будет игнорироваться. Например, если цепочка битов для точки 3 (бит 3) по указанному адресу равна 0110 и регистр сравнения цветов содержит значение 0010, то при сравнении будет возвращен байт, у которого бит 3 равен 0, если в регистре безразличия цветов все биты равны 1. Но если регистр безразличия цветов содержит 1011, то в байте, возвращаемом процессору, бит 3 будет равен 1.

Регистр безразличия цветов имеет адрес порта 3CFH и индексируется засылкой 7 в порт 3CEH. Старшие 4 его бита не играют никакой роли. Отметим, что документация IBM (от 2 августа 1984 г.) утверждает, что регистр действует обратным образом, т.е. что 1 в регистре заставляет операцию сравнения игнорировать соответствующую битовую плоскость. Эксперимент показывает обратное.

Ни один из этих двух режимов чтения не может дать быстрый ответ на вопрос о цвете определенной точки. В режиме чтения 0 необходимы 4 отдельных чтения, по одному для каждой битовой плоскости, после чего надо еще выделить соответствующие биты из каждого байта. В режиме чтения 1, с другой стороны, может потребоваться до 16 чтений, прежде чем для требуемой точки будет возвращен установленный бит, указывающий, что эта точка имеет данный цвет. Хотя EGA относительно медленно выполняет данную задачу, с другими проблемами он справляется очень быстро.

Высокий уровень

Бейсик предоставляет функцию POINT, которая возвращает цвет точки. Цвет палитры точки, находящейся в столбце 200 и строке 100, находится путем Q = POINT(200,100). Значение, возвращаемое в Q, это обычный кодовый номер цвета. Если указана точка, находящаяся за пределами экрана, то функция POINT возвращает значение -1. Когда координатная система экрана изменяется оператором WINDOW [4.4.2], функция POINT переходит к новой системе.

POINT может также сообщить позицию последней выведенной точки. При использовании обычной координатной системы, в которой 0,0 соответствует левому верхнему углу экрана, Q = POINT(1) возвращает в Q x-координату точки, а Q = POINT(2) - y-координату. Если действует оператор WINDOW, то Q = POINT(3) и Q = POINT(4) возвращают x- и y-координаты в новой системе. Когда нет активного оператора WINDOW, последние два оператора действуют так же, как и первые два.

К моменту выхода этой книги Бейсик не поддерживал улучшенные графические режимы EGA (D-10H). В этих режимах прог-

рамма должна прямо читать содержимое видеобуфера. Вот пример использования режима чтения 1 для поиска кодов цветов 0001 и 1001:

```
100 OUT &H3CE,5          'адрес регистра режима
110 OUT &H3CF,8          'устанавливаем режим чтения 0
120 OUT &H3CE,2          'адрес регистра сравнения цветов
130 OUT &H3CF,1          'ищем цвет 0001
140 OUT &H3CE,7          'адрес регистра безразличия цветов
150 OUT &H3CF,7          '7 = 0111B, поэтому ищем 0001 и 1001
160 DEF SEG = &HA000     'адрес видеобуфера для EGA
170 X = PEEK(0)          'читаем первый байт
180 IF X <> 0 THEN...    '..то цвет 0001 или 1001 найден
```

Средний уровень

Функция D прерывания 10H возвращает код цвета указанной точки. BIOS, имеющийся на плате EGA, обеспечивает работу этой функции в любом режиме дисплея. Надо поместить номер строки (отсчитываемый от 0) в DX, а номер столбца (также отсчитываемый от 0) - в CX. Результат возвращается в AL.

```
;---определяем код палитры точки 100,200
MOV  AH,0DH          ;номер функции чтения цвета точки
MOV  DX,100          ;номер строки
MOV  CX,200          ;номер столбца
INT  10H             ;теперь код цвета в AL
```

Низкий уровень

Для графических режимов цветного адаптера и PCjr надо просто обратить процесс прямого отображения в память, которым устанавливается цвет точки, как показано в [4.4.2]. Можно использовать приведенный там пример, который надо завершить следующим кодом:

```
;---изменение битов (место для вставки изменений)
MOV  AH,ES:[BX]      ;берем байт из нужной позиции
ROR  AH,CL           ;сдвигаем 2 нужных бита вниз
AND  AH,00000011B    ;выключаем остальные биты
RET                  ;теперь в AH - код палитры
```

Для режимов EGA DH - 10H надо пользоваться регистрами, описанными выше. В следующем примере режим чтения 0 используется для чтения битовой плоскости 2 по адресу A000:0012.

```
;---установка режима чтения
MOV  DX,3CEH          ;индексный регистр
MOV  AL,5             ;сначала адресуем регистр режима
OUT  DX,AL            ;посылаем индекс
INC  DX               ;указываем на сам регистр
MOV  AL,0             ;устанавливаем режим чтения 0
OUT  DX,AL
;---установка битовой плоскости, которую будем читать
DEC  DX               ;назад к индексному регистру
MOV  AL,4             ;адрес регистра выбора карты
OUT  DX,AL            ;посылаем индекс
INC  DX               ;указываем на сам регистр
MOV  AL,2             ;запрос битовой плоскости 2
OUT  DX,AL            ;посылаем значение
;---чтение битовой плоскости
MOV  AX,0A000H        ;адрес видеобуфера
MOV  ES,AX
MOV  BX,12            ;смещение в буфере
MOV  AL,ES:[BX]       ;читаем из битовой плоскости 2
```

И наконец, пример поиска кодов цвета 0010 и 1010 с использованием режима чтения 1:

```
;---установка режима чтения
MOV  DX,3CEH          ;регистр индекса
MOV  AL,5             ;адресуем сначала регистр режима
OUT  DX,AL            ;посылаем индекс
INC  DX               ;указываем на сам регистр
MOV  AL,8             ;устанавливаем бит 3 для режима 1
OUT  DX,AL            ;устанавливаем режим
;---установка регистра сравнения цветов
DEC  DX               ;возвращаемся к индексному регистру
MOV  AL,2             ;адрес регистра сравнения цветов
OUT  DX,AL            ;посылаем индекс
INC  DX               ;указываем на сам регистр
MOV  AL,0010B         ;код цвета
OUT  DX,AL            ;посылаем код
;---установка регистра безразличия цветов
DEC  DX               ;возвращаемся к индексному регистру
MOV  AL,7             ;адрес регистра безразличия цветов
OUT  DX,AL            ;посылаем индекс
```

```
INC   DX              ;указываем на сам регистр
MOV   AL,0111B        ;принимаем коды 1010 или 0010
OUT   DX,AL           ;посылаем данные
;---поиск цвета
MOV   AX,0A000H       ;адрес видеобуфера
MOV   ES,AX
MOV   BX,12           ;смещение в буфере
MOV   AL,ES:[BX]      ;читаем позицию буфера
CMP   AL,0            ;установлены биты?
JNZ   FOUND_IT        ;если да, то ищем, у какой точки
```

4.4.5. Рисование линий на экране

Простейший способ нарисовать линию на экране состоит в том, чтобы вычислить следующую точку этой линии и изменить биты соответствующего байта. Такие операции очень медленны, хотя иногда их нельзя избежать. Если это возможно, то лучше вычислить область точек экрана, которые имеют одинаковый цвет. Тогда требуемые операции над битами можно проделать только над одним байтом, а затем этот байт может быть помещен в область соответствующих позиций видеобуфера.

Высокий уровень

Бейсик позволяет рисовать прямые линии с помощью оператора LINE. LINE (20,10)-(40,30) рисует линию от столбца 20 и строки 10 к столбцу 40 и строке 30. И строки, и столбцы нумеруются от нуля. Вы можете опустить координаты первой точки, в этом случае линия будет начинаться с последней точки, которая была ранее выведена графическим оператором. Вторая пара координат может задаваться также относительно первой с помощью конструкции LINE -STEP(xoffset,yoffset).

Оператор LINE может указывать также цвет и стиль линии. Код цвета следует сразу за списком координат; LINE (50,50)-(60,60),2 выводит линию цветом 2. Когда цвет не указан, по умолчанию берется цвет 3. Возможность выбора стиля линии предполагает указание чередования ее точек. Образец может быть представлен как в десятичной, так и в шестнадцатиричной форме. Например, образец 1010101010101010, который соответствует &HAAAA, дает линию, точки которой имеют по очереди данный цвет и фоновый. Стиль линии определяется третьим параметром после координат. Например, LINE (30,30)-(40,40),3,,&HAAAA выводит линию с указанным стилем цветом 3.

Бейсик дает возможность также выполнять процедуры для рисования прямоугольников и окружностей. Прямоугольники выводятся с помощью оператора LINE. В данном случае координаты должны описывать левый верхний и правый нижний углы рамки. Надо просто указать B (box, т.е. рамка) в качестве второго параметра за координатами. LINE (50,50)-(100,100),1,B,&HAAAA рисует квадрат со стороной в 50 точек цветом 1 палитры, используя описанный выше метод. Для вывода прямоугольника, заполненного определенным цветом, надо использовать параметр BF (при этом стиль линии указывать не надо).

Окружности рисуются оператором CIRCLE. Их вывод основывается на формуле CIRCLE (x,y),r,цвет,нач-угол,кон-угол,аспект. Координаты x,y дают адрес центра окружности на экране, а r - радиус окружности в точках; вся остальная информация необязательна. Цвет - это код цвета, который по умолчанию берется равным 3. Если необходимо вывести только дугу окружности, то можно указать нач-угол и кон-угол (когда они опущены, выводится целая окружность). Углы измеряются как положительные или отрицательные величины, отсчитываемые от направления по горизонтали вправо. Они измеряются в радианах (в 360 градусах содержится 6,292 радиан, а один градус = 0,0174532 радиан). Аспект - это отношение горизонтальных и вертикальных размеров. Окружность получается на дисплее правильной формы, когда вы зададите аспект равным 5/6 для умеренного разрешения и 5/12 для высокого. Меньшие значения приводят к эллипсам, вытянутым по горизонтали, а бо́льшие - к вытянутым по вертикали. Для примера PI = 3,14159: CIRCLE(200,50),30,2,PI/2,PI,6 выводит дугу, центр которой находится в точке 50,200 с радиусом 30 точек цветом 2, причем будет выведен только левый верхний квадрант вертикально вытянутого эллипса.

Более сложные линии могут выводиться с помощью оператора DRAW, который необычайно гибок. За оператором DRAW следует строка (заключенная в скобки), в которой закодирована последовательность ориентаций и длин сегментов, составляющих линию. Например, DRAW "E12F12G12H12" выводит бубновую масть. Начальная точка устанавливается оператором PSET (обсуждаемым в [4.4.2]); в противном случае по умолчанию берется центр экрана. Основные коды состоят из буквы, за которой следует длина сегмента в точках. Коды следующие:

Ux	вверх (на x точек)
Dx	вниз
Rx	вправо
Lx	влево

Ex	по диагонали вверх и вправо
Fx	по диагонали вниз и вправо
Gx	по диагонали вниз и влево
Hx	по диагонали вверх и влево

При умеренном разрешении 100 точек по горизонтали и 100 точек по вертикали дают отрезки примерно одинаковой длины (на самом деле отношение у к х равно 5/6). При высоком разрешении горизонтальная линия будет приблизительно вдвое меньше, чем вертикальная. Из-за большего расстояния между точками диагональ прямоугольника содержит ровно столько же точек, сколько и максимальная сторона прямоугольника, хотя сам отрезок длиннее.

Для рисования диагоналей с углами, отличными от 45^0, используется кодовая буква М. Этот код рисует следующий сегмент линии в абсолютную или относительную позицию экрана. Чтобы указать абсолютную позицию, надо указать координаты х и у. DRAW "M50,60" проведет линию в точку, имеющую координаты столбца 50 и строки 60. Для указания относительных координат добавьте знаки + или - перед числами. Если текущее значение координаты х равно 100, то +50 продолжит линию до столбца 150, а -50 - до столбца 50. Чтобы сдвинуться из 100,100 в 120,70, напишите DRAW "M+20,-30".

Линия не обязательно будет непрерывной. Когда перед кодом стоит буква В, указатель перемещается заданным образом, но сегмент линии при этом не рисуется. Например, DRAW "L10BU5R10" рисует две параллельные горизонтальные линии. Чтобы из одной точки начиналось несколько сегментов, надо задать перед кодом букву N. В этом случае указатель будет возвращаться в начальную точку после вывода сегмента.

Существует ряд специальных кодов, которые, будучи помещенными внутри строки, действуют на все последующие коды (пока следующий аналогичный код не задаст другое действие). Цвет сегмента линии устанавливается буквой С, за которой следует код цвета. DRAW "C2D5" рисует линию, направленную вниз, цветом 2. Установка масштабного фактора меняет масштаб, в котором будет выводиться фигура или ее часть. Надо добавить к строке букву S, за которой следует фактор. Фактор - это число, которое для получения масштаба делится на 4. Обычно фактор равен 4, что соответствует масштабу 1:1. Изменение фактора на 8 приведет к тому, что размер выводимой фигуры будет вдвое больше. Для этого напишите DRAW "S8U12D12" и т.д.

С помощью одного из двух кодов вы можете вращать оси координатной системы. Кодовая буква А вращает оси против часовой стрелки с 90-градусными инкрементами. А0 не вращает оси

вообще. A1 поворачивает их на 90°, A2 - на 180° и A3 - на 270°. Аналогично код TA поворачивает оси на указанное число градусов от 0 до 360 (против часовой стрелки) и от 0 до -360 (по часовой стрелке). DRAW "A1L10" и DRAW "TA90L10" приведут к тому, что линия, которая должна была быть направленной влево, будет вместо этого нарисована повернутой на 90° и направлена вниз.

Оператор DRAW может включать строковые переменные, которые состоят из набора допустимых кодов. Это свойство позволяет программе повторно использовать части фигур в различных рисунках. В операторе DRAW имя строки должно быть помещено за буквой X и за ним должны следовать точка с запятой. Например:

```
100 S$ = "U12R15U45L32"
110 DRAW "XS$;"
```

В одном операторе DRAW может содержаться несколько строк, перемежаемых другими кодами. Отметим, что любые числа, используемые с кодами в операторах DRAW, могут сами быть переменными. Таким образом, с помощью одного оператора DRAW могут выводиться фигуры, отличающиеся по форме, цвету, масштабу и ориентации. Надо поместить знак равенства между буквенным кодом и именем переменной, а за именем поместить точку с запятой. Например, чтобы установить код цвета, определяемый переменной, напишите DRAW "C = PCOLOR;". Компилятор Бейсика требует, чтобы ссылка на эти переменные осуществлялась с помощью функции VARPTR$. В этом случае такой оператор будет иметь вид DRAW "X" + VARPTR$(S$) или DRAW "C = " + VARPTR$(PCOLOR). Сложные рисунки могут быть сохранены в массиве и затем возвращены на экран в любой момент. Обсуждение этого вопроса см. в [4.4.6].

Низкий уровень

В приведенной ниже процедуре применяется алгоритм Брезенхэма для вывода прямой линии, соединяющей любые две точки. В ней используется функция BIOS вывода точек и ее можно убыстрить, если заменить эту функцию на встроенную процедуру, использующую прямое отображение в память. Как и все быстрые алгоритмы, данная процедура избегает операций умножения и деления. Линия рассматривается как набор сегментов двух типов: тех, которые расположены диагонально, и тех, которые расположены горизонтально или вертикально. Для линий с наклоном больше 1 прямые сегменты вертикальны, в противном случае они горизонтальны; первая задача алгоритма состоит в вычислении

наклона. Затем вычисляется выравнивающий фактор, который следит, чтобы некоторое число прямых сегментов имело большую длину, чем остальные. И наконец, сложный цикл поочередно выводит диагональные и прямые сегменты. BX поочередно принимает то положительные, то отрицательные значения, отмечая, какой тип сегмента выводится. Ниже показано, как готовятся данные для вывода диагонали из одного угла экрана в противоположный:

```
;---в сегменте данных
ST_X     DW   0
END_X    DW   319
ST_Y     DW   0
END_Y    DW   199
COLOR    DB   2
DI_Y_IN  DW   ?
DI_X_IN  DW   ?
SH_DI    DW   ?
ST_X_IN  DW   ?
ST_Y_IN  DW   ?
ST_CO    DW   ?
DI_CO    DW   ?
;---установка режима дисплея
         MOV  AH,0              ;функция установки режима
         MOV  AL,4              ;цветной 320*200
         INT  10H               ;установка режима
;---установка начальных инкрементов для каждой позиции точки
         MOV  CX,1              ;инкремент для оси x
         MOV  DX,1              ;инкремент для оси y
;---вычисление вертикальной дистанции
         MOV  DI,END_Y          ;вычитаем координату начальной
         SUB  DI,ST_Y           ;точки из координаты конечной
         JGE  KEEP_Y            ;вперед, если наклон < 0,
         NEG  DX                ;иначе,инкремент равен -1,
         NEG  DI                ;а дистанция должна быть > 0
KEEP_Y:  MOV  DI_Y_IN,DX
;---вычисление горизонтальной дистанции
         MOV  SI,END_X          ;вычитаем координату начальной
         SUB  SI,ST_X           ;точки из координаты конечной
         JGE  KEEP_X            ;вперед, если наклон < 0,
```

```
        NEG   CX                ;иначе - инкремент равен -1,
        NEG   SI                ;а дистанция должна быть > 0
KEEP_X: MOV   DI_Y_IN,CX
;---определяем, горизонтальны или вертикальны прямые сегменты
        CMP   SI,DI             ;горизонтальные длиннее?
        JGE   HORS              ;если да, то вперед,
        MOV   CX,0              ;иначе - для прямых x не меняется
        XCHGSI,DI               ;помещаем большее в CX
        JMP   SAVS              ;сохраняем значения
HORS:   MOV   DX,0              ;теперь для прямых не меняется y
SAVS:   MOV   SH_DI,DI          ;меньшее расстояние
        MOV   ST_X_IN,CX        ;один из них 0,
        MOV   ST_Y_IN,DX        ;а другой - 1.
;---вычисляем выравнивающий фактор
        MOV   AX,SH_DI          ;меньшее расстояние в AX
        SHL   AX,1              ;удваиваем его
        MOV   ST_CO,AX          ;запоминаем его
        SUB   AX,SI             ;2хменьшее - большее
        MOV   BX,AX             ;запоминаем как счетчик цикла
        SUB   AX,SI             ;2хменьшее - 2хбольшее
        MOV   DI_CO,AX          ;запоминаем
;---подготовка к выводу линии
        MOV   CX,ST_X           ;начальная координата x
        MOV   CX,ST_Y           ;начальная координата y
        INC   SI                ;прибавляем 1 для конца
        MOV   AL,COLOR          ;берем код цвета
;---теперь выводим линию
MLOOP:  DEC   SI                ;счетчик для большего расстояния
        JZ    FINIS             ;выход после последней точки
        MOV   AH,12             ;функция вывода точки
        INT   10H               ;выводим точку
        CMP   BX,0              ;если BX < 0, то прямой сегмент
        JGE   DILIN             ;иначе - диагональный сегмент
;---выводим прямые сегменты
        ADD   CX,ST_X_IN        ;определяем инкре-
        ADD   DX,ST_Y_IN        ;менты по осям
        ADD   BX,ST_CO          ;фактор выравнивания
        JMP   SHORT MLOOP       ;на следующую точку
;---выводим диагональные сегменты
```

```
DILIN:  ADD  CX,DI_X_IN ;определяем
        ADD  DX,DI_Y_IN          ;инкременты по осям
        ADD  BX,DI_CO            ;фактор выравнивания
        JMP  SHORT MLOOP         ;на следующую точку
FINIS:
```

4.4.6. Заполнение областей экрана

Тщательное продумывание позволяет исключить излишнюю медлительность, свойственную многим программам заполнения областей для графического экрана. Когда заполнение основано на простых вычислениях, которые действуют по очереди для каждой точки, требуются расходующие много времени битовые операции. Более экономный код может определять, все ли битовые позиции определенного байта видеобуфера должны иметь один и тот же цвет, и когда это условие выполняется, этому байту присваивается заранее заготовленное значение, которое устанавливает все точки в правильный цвет. При этом нет необходимости повторять операции над одним и тем же байтом, каждый раз устанавливая биты только для одной из точек, информацию о которой содержит данный байт.

В [4.3.4] объяснено, как создать описание символа в виде матрицы 8*8 точек, имеющего требуемый вам вид. Хотя такие символы могут выводиться только в стандартные символьные позиции, их использование может существенно облегчить заполнение графиков. Образец, высвечивающий все 8*8 точек можно вывести в интервале нескольких строк и столбцов, заполняя область намного быстрее, чем это достигается при поточечной зарисовке. Этот тип графических символов может применяться совместно с точечной графикой. Псевдографические символы могут служить также для вывода вращающихся или колеблющихся объектов.

Высокий уровень

Бейсик предоставляет оператор PAINT для заполнения замкнутой фигуры произвольной формы. Вам необходимо указать только точку внутри области, а об остальном позаботится процедура. Может быть указан цвет палитры, которым надо заполнить область, например PAINT (100,110),2 заполняет область цветом 2 палитры. Закраска ведется начиная от указанной точки до тех пор, пока не встретятся точки с цветом, отличающимся от фонового. Вы можете, наоборот, указать цвет границы, и закраска

будет продолжаться во всех направлениях, пока не будут встречены точки указанного цвета. При такой закраске линии других цветов, находящиеся внутри границы, могут быть также закрашены. Код цвета границы следует за кодом цвета заполнения, таким образом, PAINT (100,180),2,3 закрашивает область цветом 2 до линий цвета 3. Отметим, однако, что эта процедура не заполняет области, находящиеся "за углом", т.е. если вдоль какой-либо горизонтальной или вертикальной траектории встретилась точка, имеющая цвет границы, все последующие точки вдоль этой траектории не заполняются, даже когда фигура имеет причудливую форму и эти точки принадлежат внутренней части фигуры. В следующем примере выводятся две перекрывающиеся рамки цветами циан и магента, а затем последняя рамка заполняется белым цветом. Сегменты первой рамки, которые попадают в закрашенную область, также заполняются белым.

```
100 LINE (50,70)-(270,130),1,B          'рисуем рамку цветом циан
110 LINE (100,30)-(220,170),2,B         'рисуем рамку цветом магента
120 PAINT (101,31),3,2                  'заполняем вторую рамку белым
```

Помните, что команда LINE может сама заполнить рамку, если вы укажете в качестве параметра 'BF', а не 'B', см. [4.4.5].

Оператор PAINT имеет "орнаментальные" возможности, которые позволяют заполнять области указанной картинкой. Элементы орнамента, которые в режиме умеренного разрешения имеют размер 4 точки в ширину и 8 в высоту (8*8 для высокого разрешения), повторяются по всей указанной области. Рисунок описывается набором байтов, содержащих цепочку битов для последовательных рядов элемента орнамента. В режиме умеренного разрешения цепочка битов 10000011 описывает 4 точки, первая из которых имеет цвет 2, следующие 2 - фоновый цвет, а последняя - цвет 3. Эта цепочка соответствует числу 131 или &H83 (см. приложение Б, где обсуждаются битовые операции в Бейсике). Обращение этой цепочки в 11000010 даст 193 (&HC1). Они могут быть объединены в элемент орнамента шириной в 4 точки и высотой в 2 точки строкой CHR$(&H83) + CHR$(&HC1). В такую строку можно включать до 8 байт, доводя высоту до 8 точек. Такая строка используется в операторе PAINT вместо цвета. Вот вывод квадрата, заполненного описанным орнаментом:

```
100 LINE (100,110)-(150,150),1,B                'рисуем рамку
110 PAINT (125,125),CHR$(&H83) + CHR$(&HC1),1   'заполняем ее
```

Отметим, что нерегулярности элемента орнамента могут приводить к тому, что процедура PAINT завершается, не закончив заполнения области. Бейсик решает эту проблему указанием параметра фона для оператора PAINT. Если у вас возникнут проблемы, обращайтесь к руководству по Бейсику за деталями.

Оператор DRAW, позволяющий рисовать сложные линии, также может заполнять области. Он обсуждается в [4.4.5]. "Текущая точка" (из которой будет рисоваться следующий сегмент линии) должна быть помещена внутрь области, ограниченной линией указанного цвета. В строку оператора DRAW надо поместить кодовую букву P, за которой должен следовать код цвета закраски и код цвета границы. Для вывода рамки цветом 1 палитры, а затем для ее заполнения цветом 3 напишите DRAW "U10R10D10L10BH1P3,1". Здесь первые четыре кода рисуют границы рамки, затем код 'BH' перемещает текущую точку внутрь рамки, не рисуя линии, а затем код 'P' приводит к заполнению рамки. Таким образом могут быть заполнены и более сложные формы. Отметим, что необязательно при перемещении точки внутрь области отменять рисование линии вдоль этого пути. Однако в данном случае надо использовать для этого сегмента код цвета, отличный от цвета заполняемой границы.

Бейсик имеет также возможность заполнения областей экрана заранее подготовленным изображением. Изображение может быть любого размера, может быть выведено в любой позиции экрана и храниться в массиве. Обычно изображение создается с помощью всех доступных средств, а затем запоминается в массиве оператором GET. Массив может быть помещен в последовательный файл [5.4.3], из которого программа может загрузить его и вывести изображение. Оператор GET перечисляет координаты левого верхнего и правого нижнего угла рамки, содержащей изображение, причем сначала идет номер столбца, а затем номер строки для каждой пары координат. Затем должно следовать имя массива, которое не заключается в кавычки. Например, GET (80,40)-(120,60),ARRAY3 помещает все точки, находящиеся внутри указанной области, в массив с именем ARRAY3.

Одномерные массивы, как и все остальные, должны быть предварительно описаны оператором DIM. Массив может содержать элементы любой точности. Для вычисления требуемых размеров массива надо сначала определить, сколько байтов потребуется для хранения изображения. Это можно вычислить по формуле 4 + INT ((Xхбитовнаточку + 7)/8)x Y. Здесь "битовнаточку" = 1 для высокого разрешения и 2 - для умеренного. Буквы X и Y относятся к числу точек вдоль горизонтальной и вертикальной сторон блока изображения. INT обозначает целую часть числа. Наконец, надо определить, сколько элементов

массива требуется для хранения данного числа байтов. Каждый элемент занимает 2 байта в целом массиве, но 4 для чисел с обычной точностью и 8 - для чисел с двойной точностью.

Для получения изображения из массива и вывода его на экран служит оператор PUT. Этот оператор требует только координаты левого верхнего угла области экрана, в которую будет выводиться изображение. За координатами должно быть указано имя массива. Например, PUT (40,30),ARRAY1 помещает изображение, левый верхний угол которого будет находиться в столбце 40 и строке 30. Оператор PUT может иметь еще и необязательный параметр, определяющий цвет, которым будет выводиться изображение. Если этот параметр опущен, то изображение будет выводиться точно в том виде, в котором оно было записано оператором GET. Это эквивалентно записи PUT (40,30),ARRAY1,PSET. В противном случае имеются некоторые другие возможности. Если вы вместо PSET укажете PRESET, то цвет 0 палитры будет заменен на цвет 3 и наоборот, а цвет 1 палитры - на цвет 2 и наоборот.

Существуют еще три ситуации с логическими операциями AND, OR или XOR. Как и PRESET, эти слова могут заменять PSET в приведенном примере. Обсуждение этих трех операций содержится в приложении Б. Каждая операция включает сравнение битов существующей точки на экране с битами точки накладываемого изображения. В режиме высокого разрешения, когда на точку отводится только 1 бит, операция простая. Но в режиме умеренного разрешения, в котором на каждую точку отводится по 2 бита, могут происходить различные трансформации цветов.

AND устанавливает бит, только если он был установлен и у точки экрана, и у точки изображения (взятой из массива). В режиме высокого разрешения это означает, что точка изображения появится на экране, только если соответствующая точка экрана уже "включена". Все остальные точки области будут выключены. В режиме умеренного разрешения операция производится над обоими битами. Если для точки экрана установка битов 01, а для соответствующей точки изображения - 10, то оба бита будут сброшены и точка экрана получит код 00, что соответствует фоновому цвету.

OR устанавливает бит, если он был установлен либо для точки экрана, либо для точки изображения. В черно-белом режиме OR накладывает изображение на существующее изображение на экране. В цветном режиме для определения эффекта вы снова должны прибегнуть к вычислениям. Комбинация кодов палитры 1(01) и 2(10) дает 3(11), так же как и комбинация 0(00) и 3(11).

И наконец, XOR устанавливает бит, если из двух сравниваемых только один был установлен. Применение этой операции

для черно-белого экрана с массивом единиц обеспечивает негативное изображение (1 и 1 дает 0, а 1 и 0 - 1). В режиме умеренного разрешения эта операция меняет все цвета. В результате получаем наложение двух изображений. Но более важно, что при повторении этой операции экран принимает в точности такой же вид, который он имел первоначально. При этом изображение стирается. Такая методика полезна для мультипликации, когда над изображением дважды производится операция XOR в одной позиции, затем в соседней и т.д.

Низкий уровень

Существует много подходов к написанию процедур заполнения графических объектов. Ни один из них не является идеальным, поскольку всегда возможен конфликт между скоростью работы процедуры и сложностью фигур, которые она может обрабатывать. Любая процедура, заполняющая область точка за точкой, будет медленной независимо от того, насколько элегантно она реализована. Имейте в виду, что почти каждая модифицируемая точка расположена в байте, все точки которого будут изменяться в тот же цвет. Получение доступа к одному и тому же байту с использованием сложных процедур требует существенно больше времени, чем установка целого байта за один доступ к ячейке видеобуфера. Например, поточечная очистка экрана требует на IBM PC нескольких секунд при применении функции BIOS, в то время как прямой доступ в память производит эту операцию мгновенно:

```
MOV  AX,0B800H      ;ES указывает на буфер экрана
MOV  ES,AX
MOV  CX,8192        ;заполняем все байты
MOV  AX,0           ;в каждый байт пишем 0
MOV  DI,0           ;DI поочередно указывает на все байты
REP  STOSW          ;повторяем запись 8192 раза
```

Многие процедуры заполняют по одной горизонтальной строке, проверяя на цвет границы справа и слева. Поскольку строки состоят из смежных байтов данных, надо поочередно брать байты из видеобуфера и проверять, присутствует ли в них цвет границы. Если цвет границы отсутствует, то можно заменить сразу весь байт на цвет заполнения. В противном случае к данному байту применяется поточечный подход.

Есть очень быстрый способ определения, присутствует ли граничный цвет в данном байте видеобуфера. Предположим, что процедура ищет цвет 1 палитры в режиме умеренного разрешения с четырьмя цветами. Этому цвету соответствует код 01, поэтому

сначала заполним весь байт этим кодом: 01010101. Затем воспользуемся операцией NOT для обращения каждого бита, после чего байт примет вид 10101010. Проделаем операцию XOR со значением, взятым из видеобуфера; в результате получим байт, у которого оба бита, относящиеся к одной точке, равны 1 только для точек, имеющих граничный цвет. Затем снова применим операцию NOT с тем, чтобы пара битов, относящихся к точке граничного цвета, имела код 00. После этого используем операцию TEST для нахождения полей со значением 00. Если такое поле найдено, то граничный цвет обнаружен и процедура переходит к обычному поточечному анализу данного байта. Эту процедуру можно еще ускорить, если использовать словные данные.

```
MOV   AL,ES:[BX]        ;берем байт из видеобуфера
XOR   AL,10101010B      ;устанавливаем биты для цвета границы
NOT   AL                ;обращаем биты
TEST  AL,11000000B      ;проверяем биты 7
JZ    FOUND_BOUND       ;переход, если граничный цвет
TEST  AL,00110000B      ;проверяем биты 5
JZ    FOUND_BOUND       ;переход, если граничный цвет
TEST  AL,00001100B      ;проверяем биты 3
JZ    FOUND_BOUND       ;переход, если граничный цвет
TEST  AL,00000011B      ;проверяем биты 1
JZ    FOUND_BOUND       ;переход, если граничный цвет
MOV   AL,FILL_COLOR     ;граничного цвета нет, заполняем байт
MOV   ES:[BX],AL        ;возвращаем байт в видеобуфер
.
.
FOUND_BOUND:
```

Когда это возможно, постарайтесь, чтобы границы прямоугольных областей ваших картинок были выравнены на границу двух, четырех или восьми точек с тем, чтобы прямое отображение в память имело дело с целыми байтами. Другая возможность, хотя и при меньшей скорости, состоит в создании определяемых пользователем псевдографических символов [4.3.4] и выводе их на границе области заполнения. Короче, в данной области вы всегда можете проявить сообразительность, а зачастую стоит подумать, нужна ли вам столь сложная графика в данной задаче.

4.4.7. Графический вывод с использованием символов псевдографики

Когда вы выводите изображение точка за точкой, это отнимает очень много времени, особенно когда создаются эффекты мульти-

пликации. Один из способов экономии времени состоит в сведении всех или части выводимых форм к фигурам, которые могут быть построены на матрице точек 8*8. Такие фигуры могут быть созданы как определяемые пользователем символы, см. [4.3.4]. После того как эти символы определены, они выводятся на экран очень быстро и просто. Эти символы могут выводиться вперемешку с поточечными графиками, как обычные буквы. Один из способов быстрого заполнения фигуры состоит в последовательном выводе внутри фигуры полностью закрашенного блока. Отметим, что эти символы всегда располагаются в стандартных позициях курсора.

Средний уровень

В этом примере рисуется фигура человека, занимающая 2 символа в высоту и 2 символа в ширину. Как объяснено в [4.3.4], вектор прерывания 1FH указывает на начало области данных, определяющих символы. Четыре символа могут быть выведены обычными процедурами DOS или BIOS. Легко создать набор символов для вывода фигуры с руками и ногами в другом месте экрана. Два набора символов могут поочередно меняться в соседних позициях курсора, создавая иллюзию идущего по экрану человека.

```
;---в сегменте данных
CHAR_DAT DB  00110000B      ;левый верхний квадрант
         DB  01100111B
         DB  01100111B
         DB  00110011B
         DB  00011111B
         DB  00001111B
         DB  00001111B
         DB  00000111B
         DB  00000011B      ;правый верхний квадрант
         DB  10001100B
         DB  10011000B
         DB  00110000B
         DB  11100000B
         DB  11000000B
         DB  11000000B
         DB  10000000B
         DB  00001111B      ;левый нижний квадрант
         DB  00011111B
```

```
                DB  00011100B
                DB  00011000B
                DB  00011000B
                DB  00110000B
                DB  01100000B
                DB  00010000B
                DB  11000000B       ;правый нижний квадрант
                DB  11000000B
                DB  11000000B
                DB  11000000B
                DB  01100000B
                DB  01100000B
                DB  00010000B
                DB  00011100B
                DB  00000000B
;---установка вектора прерывания
PUSH DS                 ;сохраняем DS
MOV  DX,OFFSET CHAR_DAT ;смещение для данных в DX
MOV  AX,SEG CHAR_DAT    ;сегмент для данных в DS
MOV  DS,AX
MOV  AH,25H             ;функция установки вектора
MOV  AL,1FH             ;номер вектора
INT  21H                ;устанавливаем вектор
POP  DS                 ;восстанавливаем DS
;---рисуем фигуру
;---позиционируем курсор на верхний ряд
MOV  AH,2               ;функция установки курсора
MOV  DH,13              ;строка 13
MOV  DL,20              ;столбец 20
MOV  BH,0               ;страница 0
INT  10H                ;установка курсора
;---рисуем верхние два символа
MOV  DL,128             ;берем символ 128
MOV  AH,2               ;функция вывода/курсор вперед
INT  21H                ;вывод символа
MOV  DL,129             ;берем символ 129
INT  21H                ;выводим его
;---позиционируем курсор на нижнюю строку
MOV  DH,14              ;строка 14
```

```
MOV  DL,20          ;столбец 20
MOV  AH,2           ;функция установки курсора
INT  10H            ;устанавливаем курсор
;---рисуем нижние два символа
MOV  DL,130         ;берем символ 130
MOV  AH,2           ;функция вывода/курсор вперед
INT  21H            ;вывод символа
MOV  DL,131         ;берем символ 131
INT  21H            ;выводим его
```

4.5. Сдвиг экрана и страницы

Сдвиг экрана и разбиение на страницы - это два способа переноса блока информации из памяти на экран. При сдвиге одна из границ экрана сдвигается внутрь, стирая информацию на противоположной стороне. Затем освободившаяся область заполняется из памяти. Повторение этого действия строка за строкой создает иллюзию сдвига экрана.

С другой стороны, разбиение на страницы основано на одновременном хранении нескольких экранов информации в видеобуфере и переключении вывода с одной страницы на другую. Использование дисплейных страниц невозможно на монохромном адаптере, поскольку его памяти хватает только для одного символьного экрана. Другие видеосистемы в большинстве экранных режимов могут работать с несколькими страницами. Использование страниц дисплея особенно полезно при построении сложных картин "за кулисами"; после того как эта работа завершена, новый экран выводится моментально. Процедура, имитирующая работу со страницами для монохромного адаптера, приведена в [4.5.3]. Она особенно полезна, когда вы имеете дело с медленным выводом на экран в Бейсике.

4.5.1. Вертикальный сдвиг текстового экрана

Когда текстовый экран сдвигается вверх, строки со 2-й по 25-ю переписываются на строки с 1-й по 24-ю, а следующая строка данных выводится в 25-й строке. При этом верхняя строка, поверх которой осуществляется вывод, теряется, хотя она продолжает существовать в памяти. Сдвиг вниз устроен аналогично.

Высокий уровень

Бейсик утомительно медлителен при манипуляциях с экраном. Для быстрого сдвига вы можете воспользоваться процедурой на машинном языке, которая не делает ничего другого, кроме прерывания 10H, см."Средний уровень". Процедура позволяет сдвигать весь экран или любое окно в нем. В приложении Г показано, как включать подпрограммы на машинном языке в программы. Ваша программа на Бейсике должна указывать координаты верхнего левого и нижнего правого углов окна, которые могут лежать в диапазоне от 0 до 24 и от 0 до 79. Требуется также параметр, указывающий направление сдвига: вверх или вниз (6 и 7 соответственно), число строк, на которое нужно сдвинуть (если 0, то окно очищается), и значение байта атрибутов для очищаемых строк (для "нормальных" - 7). Используйте для них целые переменные. В приведенном примере экран сдвигается вниз на одну строку, а затем освободившаяся строка заполняется.

```
100 '''данные для подпрограммы
110 DATA &H55, &H8B, &HEC, &H8B, &H76, &H12, &H8A
120 DATA &H24, &H8B, &H76, &H10, &H8A, &H04, &H8B
130 DATA &H76, &H0E, &H8A, &H2C, &H8B, &H76, &H0C
140 DATA &H8A, &H0C, &H8B, &H76, &H0A, &H8A, &H34
150 DATA &H8B, &H76, &H08, &H8A, &H14, &H8B, &H76
160 DATA &H06, &H8A, &H3C, &HCD, &H10, &H5D, &HCA
170 DATA &H0E, &H00
180 '''помещаем данные в сегмент &H2000
190 DEF SEG = &H2000                    'помещаем данные начиная с &H2000
200 FOR N = 0 TO 43                     '44 байта
210 READ Q                              'читаем один байт
220 POKE N,Q                            'помещаем его в память
230 NEXT                                'следующий
300 '''в программе
310 GOSUB 500                           'сдвигаем на строку
320 LOCATE 1,1: PRINT TEXT$(LINP);'выводим строку текста
500 '''подпрограмма сдвига
510 DEFINT A-Z                          'используем целые переменные
520 TLR = 0                             'левая верхняя строка
530 TLC = 0                             'левый верхний столбец
540 BRR = 24                            'нижняя правая строка
550 BRC = 79                            'нижний правый столбец
```

```
560 NUMROWS = 1                          'число строк сдвига
570 DIR = 7                              'направление сдвига вниз
580 FILL = 7                             'заполнение обычным атрибутом
590 DEF SEG = &H2000                     'указываем на подпрограмму
600 SCROLL = 0                           'начинаем с 1-го байта
610 CALL SCROLL(DIR,NUMROWS,TLR,TLC,BRR,BRC,FILL)
620 RETURN                               'все сделано
```

Средний уровень

Функция 6 прерывания 10H сдвигает любую часть экрана вверх, а функция 7 - вниз. В обоих случаях AL содержит число строк сдвига, а когда AL = 0, весь экран чистится, а не сдвигается. CH:CL содержат строку и столбец левого верхнего угла, а DH:DL - координаты правого нижнего угла. Появляющиеся из-за сдвига строки - чистые; они выводятся с кодом атрибутов из BH.

```
;---сдвиг вверх на одну строку
MOV  AH,6          ;номер функции сдвига вверх
MOV  AL,1          ;число строк сдвига
MOV  CH,0          ;строка левого верхнего угла
MOV  CL,0          ;столбец левого верхнего угла
MOV  DH,24         ;строка правого нижнего угла
MOV  DL,79         ;столбец правого нижнего угла
MOV  BH,7          ;атрибуты очищаемой строки
INT  10H           ;делаем сдвиг
```

Низкий уровень

Вертикальный сдвиг всего экрана - тривиальная задача, поскольку правая граница одной строки в памяти продолжается левой границей следующей строки. Сдвиг всего содержимого видеобуфера на 160 байт вверх по памяти (80 символов в строке x 2 байта на символ) приводит к сдвигу экрана вниз на одну строку. Если вы пишете свою собственную процедуру сдвига экрана, использующую прямое отображение в память, то не забывайте об интерференции, которая возникает на цветном дисплее и PCjr. Эта проблема обсуждается в [4.3.1]. Решение этой проблемы - проверять статусный байт, ожидая, пока он разрешит запись в видеобуфер. Вам придется поэкспериментировать, чтобы определить, сколько данных вы можете записать за один цикл.

Другое решение состоит в выключении экрана на время операции сдвига, а затем в его восстановлении. "Выключение экрана" подразумевает, что вывод содержащихся в видеобуфере данных запрещен, но сам буфер не изменяется. Описанный процесс пользуется функцией сдвига BIOS. Это не очень приятно для глаз, но все-таки не так плохо, как уже упоминавшаяся интерференция.

Для выключения экрана у цветного графического дисплея надо сбросить бит 3 порта с адресом 3D8H. Изменение бита назад на 1 моментально вновь включает экран. Адрес порта соответствует регистру выбора режима цветного графического адаптера. Этот однобайтовый регистр только для записи, поэтому программа не может просто прочитать его, изменить значение бита 3 и вернуть прочитанный байт. Вместо этого необходимо определить также правильную установку всех остальных битов (перечисленных в [4.1.2]). Для PCjr этот бит расположен в регистре управления режимом 1 массива ворот дисплея. В [4.1.1] объяснено, как получить доступ и запрограммировать этот регистр.

4.5.2. Сдвиг текстового экрана горизонтально

Горизонтальный сдвиг иногда требуется в специальных программах обработки текста, таких, как текстовые редакторы. Операционная система не имеет для этого специальных средств. По этой причине данная задача немного сложнее, чем вертикальный сдвиг. Рассмотрим ситуацию, когда вы хотите, чтобы экран сдвигался влево на 5 позиций. При этом левые 5 столбцов исчезнут, весь остальной текст сдвинется влево, а самые правые 5 столбцов будут очищены. Видеобуфер представляет собой одну длинную строку. Поэтому если каждый символ буфера сдвинуть на 10 байт вниз, то суммарный эффект будет состоять в том, что самые левые 5 символов каждой строки будут передвинуты в последние 5 позиций предыдущей строки. Таким образом, весь экран будет сдвинут влево на 5 позиций, передвигая 5 ненужных столбцов в правую часть экрана. Все, что после этого остается, - это очистить правые 5 столбцов. Это легко делается с помощью процедуры вертикального сдвига [4.5.1], которая может выполняться для любой части экрана и которая очищает указанную область, если указать сдвиг на 0 строк. Рис. 4.6 иллюстрирует описанный метод.

Низкий уровень

В этом примере осуществляется сдвиг на 5 позиций влево. Легко изменить его для сдвига вправо или для другого значения

позиций сдвига. При использовании прямого отображения в память этот метод дает практически моментальный сдвиг экрана.

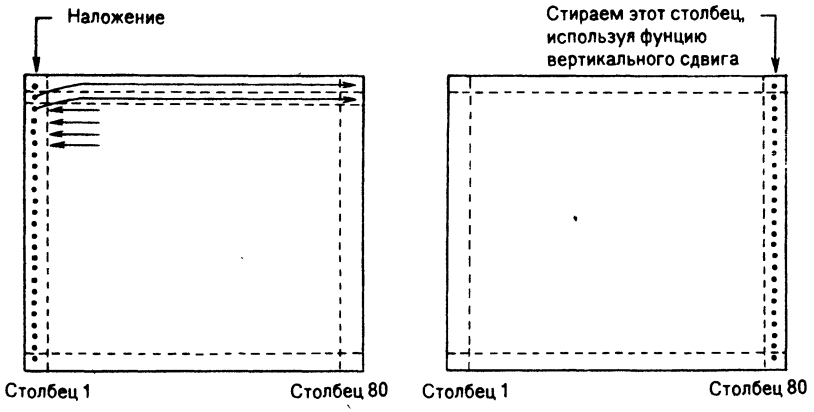


Рис. 4.6. Горизонтальный сдвиг экрана

```
;---сдвигаем все вниз на 10 байт
        MOV AX,0B000H   ;указываем на буфер монохромного
        MOV ES,AX       ;дисплея
        MOV DS,AX
        MOV SI,10       ;сдвигаем из SI ...
        MOV DI,0        ;... в DI
        MOV CX,1995     ;сдвигаем все, кроме последних 5 байт
REP     MOVSW           ;осуществляем сдвиг
;---очищаем правый край
        MOV AH,6        ;функция вертикального сдвига
        MOV AL,0        ;сдвиг на 0 строк чистит окно
        MOV CH,0        ;строка левого верхнего угла
        MOV CL,75       ;столбец левого верхнего угла
        MOV DH,24       ;строка правого нижнего угла
        MOV DL,79       ;столбец правого нижнего угла
        MOV BH,7        ;атрибут для очищаемых позиций
        INT 10H         ;чистим окно
```

4.5.3. Переключение между текстовыми страницами

Поскольку все видеосистемы, кроме монохромного дисплея, имеют достаточно памяти для нескольких видеобуферов, одновременно могут быть сконструированы несколько экранов, каждый из которых может быть выведен в нужный момент. Вместо того чтобы передвигать данные в видеопамяти, монитор посылает данные из другой области видеопамяти. Число доступных страниц может меняться в зависимости от видеосистемы и режима дисплея. Приводим краткую сводку:

Режим	Тип	Число страниц	Начало буфера
0	алфавитно-цифровой	8	B800
1	алфавитно-цифровой	8	B800
2	алфавитно-цифровой	8	B800
3	алфавитно-цифровой	8	B800
4	графический	1	B800
5	графический	1	B800
6	графический	1	B800
7	алфавитно-цифровой	1/8	B800
8	графический	переменное	B800
9	графический	переменное	B800
A	графический	переменное	B800
D	графический	2/4/8	A000
E	графический	1/2/4	A000
F	графический	1/2	A000
10	графический	1/2	A000

Режимы 8-A - графические режимы PCjr; число страниц для них меняется в зависимости от того, сколько оперативной памяти отведено под видеобуфер. Размер страницы равен 2K или 4K для алфавитно-цифровых режимов, 32K - для четырех цветов при высоком разрешении или 16 цветов при умеренном разрешении и 16K - для всех остальных режимов. Режимы D-10 поддерживаются EGA. Количество страниц меняется в зависимости от установленной памяти. Режимы F и 10 требуют наличия не менее 128K памяти. Режим 7 разрешает одну страницу для монохромного адаптера и 8 страниц для EGA.

Монохромный адаптер не имеет памяти для дополнительных страниц. Однако нет никаких причин, по которым часть основной памяти нельзя было бы использовать как буфер дисплея. В этом случае страничная организация осуществляется за счет быстрого обмена всего содержимого буфера в памяти с видеобуфером (адрес которого B000:0000). Буфер в основной памяти можно рассмат-

ривать как "псевдостраницу". Хотя это и не настоящее разбиение на страницы, результат будет почти такой же, если для пересылки данных вы будете использовать ассемблерную процедуру.

При использовании страниц надо позаботиться о том, чтобы операции вывода на экран направлялись на нужную страницу. Программа не обязана выводить данные на ту страницу, которая в данный момент изображается на экране. На самом деле часто желательно конструировать экран "за кулисами", а затем моментально выводить готовое изображение. Этот метод особенно полезен, когда необходимо конструировать сложный вывод в Бейсике, у которого вывод очень медленный. BIOS хранит в своей области данных однобайтовую переменную, указывающую, какая из страниц выводится в данный момент. Диапазон значений этой переменной от 0 до 7. Она расположена по адресу 0040:0062.

Высокий уровень

В Бейсике предусмотрена команда SCREEN для установки страницы, на которую будет идти вывод (активной страницы), и выводимой страницы (видимой страницы). Страницы нумеруются от 0 до 3 для текстов с 80 символами в строке и от 0 до 7 для 40-символьных. Третий параметр за командой SCREEN устанавливает активную страницу. SCREEN,,2 приводит к тому, что все операторы PRINT будут работать со страницей 2. Четвертый параметр устанавливает видимую страницу. SCREEN,,,1 приводит к тому, что на экран будет выводиться страница 1. Если указать SCREEN,,2,1 - то вывод будет направлен на 2 страницу, а видимой будет страница 1. Когда видимая страница не указывается, автоматически принимается, что она совпадает с активной.

Для выделения памяти под страницы на PCjr используется оператор CLEAR. Этот оператор устанавливает общее количество памяти, отводимое под буфер экрана, которое при старте равно 16384 байтам. Чтобы добавить вторую страницу размером 16K, напишите CLEAR,,32768. Добавочные текстовые страницы требуют 4096 байт каждая. При условии, что таким способом была отведена память, команды оператора SCREEN для работы со страницами будут действовать ранее описанным образом. Только PCjr имеет добавочный параметр оператора SCREEN, который стирает страницу (т.е. переводит ее в цвет фона). Детали описаны в руководстве по Бейсику. Оператор PCOPY также уникален для PCjr. Он копирует изображение из одной страницы в другую. Например, PCOPY 2,1 целиком копирует страницу 2 на страницу 1.

Хотя монохромный адаптер не имеет памяти для страниц дисплея, существует способ устроить своего рода "псевдостраницы". Приведенная ниже процедура на машинном языке рас-

сматривает блок памяти как дисплейную страницу. При вызове этой процедуры она обменивает содержимое видеобуфера с содержимым этой области памяти. В результате мы имеем как бы две дисплейные страницы. (В приложении Г объясняется, как включать подпрограммы на машинном языке в программы на Бейсике.)

Вы должны отвести блок памяти размером 4000 байт для псевдостраницы, помимо памяти, содержащей программу на машинном языке. В примере блок начинается с адреса сегмента &H2000, а процедура помещена по адресу &H2200. Сегментный адрес блока содержится в 9-м и 10-м байтах машинного кода, и вы легко можете изменить его. Видно, что адрес &H2000 представлен как &H00, &H20 в операторе DATA. Это следствие того, что младшие цифры всегда размещаются в младших ячейках памяти. Если вы хотите разместить блок, скажем по адресу 1234:0000, то надо изменить байты 9 и 10 на &H34, &H12.

Вам может потребоваться очистить псевдостраницу от всякой ерунды, оставшейся от других программ. В строках 230-260 это достигается за счет засылки символа пробела (ASCII 32) в каждый байт (32 служит "нормальным" байтом атрибутов). Программа может осуществлять вывод на экран обычным образом, а затем переносить содержимое на псевдостраницу. Но при желании вы можете осуществлять вывод непосредственно на псевдостраницу, используя прямое отображение в память.

```
100 '''машинный код
110 DATA &H1E, &H06, &HB8, &H00, &HB0, &H8E, &HC0
120 DATA &HB8, (3&H00, &H20), &H8E, &HD8, &HBF, &H00
130 DATA &H00, &HBE, &H00, &H00, &HFC, &HB9, &HD0
140 DATA &H07, &H26, &H8B, &H1D, &HAD, &HAB, &H89
150 DATA &H5D, &H1E, &HE2, &HF6, &H07, &H1F, &HCB
160 '''помещаем код в память
170 DEF SEG = &H2200          'указываем адрес процедуры
180 FOR N = 0 TO 34           'начинаем с первого байта
190 READ Q                    'читаем байт процедуры
200 POKE N,Q                  'пишем его в память
210 NEXT
220 '''чистим псведостраницу
230 DEF SEG = &H2000          'адрес начала псевдостраницы
240 FOR N = 0 TO 3999         'для каждого символа и атрибута
250 POKE N,32                 'помещаем код 32
260 NEXT                      'пока не очистим весь буфер
```

```
500 '''пишем прямо в псевдостраницу
510 DEF SEG = &H2000              'указываем на ее адрес
520 S$ = "PSEUDOPAGE"             'выводим слово посреди страницы
530 M = LEN(S$)                   'получаем длину строки
540 FOR N = 1 TO M                'для каждого символа строки
550 POKE N*2 + 2000, ASC(MID$(S$,N,1)) 'помещаем его в буфер
560 NEXT

600 '''теперь используем процедуру
610 PRINT "SCREEN 1"              'печатаем сообщение на экран
620 DEF SEG = &H2200              'указываем на процедуру
630 PSEUDOPAGE = 0                'начинаем с начала процедуры
640 CALL PSEUDOPAGE               'обмениваем страницы
650 CALL PSEUDOPAGE               'повторяем обмен
660 ...
```

Средний уровень

Функция 5 прерывания 10H выбирает текущую страницу дисплея для вывода. Надо просто поместить номер страницы в AL:

```
;---установка видимой страницы
MOV  AH,5          ;номер функции
MOV  AL,2          ;номер страницы (начиная с 0)
INT  10H           ;устанавливаем страницу
```

Однако эта функция не устанавливает страницу, на которую будет идти вывод. Любое из прерываний BIOS, которые выводят на экран (функции прерывания 10H), требует, чтобы номер страницы был указан в качестве входного параметра в одном из регистров. Однако все прерывания вывода на экран MS-DOS пишут на текущую видимую страницу. Таким образом, для "закулисных" операций необходимо пользоваться прерыванием 10H.

Для получения информации о текущей странице надо выполнить функцию F прерывания 10H, которая возвращает статус дисплея. Номер страницы при этом возвращается в BH.

Низкий уровень

Дисплейные страницы выбираются за счет изменения точки видеопамяти, начиная с которой монитор принимает данные. Эта

точка памяти устанавливается регистрами 12 (старший байт) и 13 (младший байт) микросхемы 6845, которые называются регистрами стартового адреса. Значения адресов раздела страниц для буфера, начинающегося с B800, такие:

Страница	40 символов	80 символов
0	0000H	0000H
1	0400H	0800H
2	0800H	1000H
3	0C00H	1800H
4	1000H	
5	1400H	
6	1800H	
7	1C00H	

В [4.1.1] объясняется, как программировать регистры микросхемы 6845, а в [4.5.4] содержится пример программирования стартового адреса. В последнем примере надо просто присвоить BX одно из значений приведенной таблицы. Конечно, при этом устанавливается только выводимая страница. Для записи в определенную страницу на низком уровне надо использовать одно из значений таблицы в качестве смещения в видеобуфсре при прямом отображении в память.

Поскольку прямое отображение в память работает очень быстро, иллюзия страниц может быть легко создана на монохромном дисплее. Выделите блок размером 4000 байт для хранения страницы. Хотя монохромный адаптер не может непосредственно читать из обычной памяти, содержимое этого буфера и видеобуфера можно обменять настолько быстро, что никто не заметит разницы. Следующая процедура обменивает содержимое этих двух областей.

```
;---в сегменте данных
PPAGEDW    2000 DUP(720H)      ;заполняем буфер пробелами
;---пересылка между псевдостраницей и видеобуфером
        MOV   AX,0B000H        ;указываем на видеобуфер
        MOV   ES,AX
        MOV   AX,SEG PPAGE     ;указываем на псевдостраницу
        MOV   DS,AX
REPE:   MOV   DI,0             ;DI на начало видеобуфера
        MOV   SI,OFFSET PPAGE  ;SI на начало псевдостраницы
```

```
        CLD                     ;направление вперед
        MOV   CX,2000           ;будем пересылать 2000 слов
NEXT:   MOV   BX,ES:[DI]        ;берем слово из видеобуфера в BX
        LODSW                   ;слово из псевдостраницы в AX
        STOSW                   ;слово из AX в видеобуфер
        MOV   DS:[DI]-2,BX      ;слово из BX в псевдостраницу
        LOOP  NEXT
```

PCjr хранит регистр страницы в порте с адресом 3DFH. Значение битов этого регистра следующее:

биты 2-0 какая страница выводится (от 0 до 7)
5-3 какая страница пишется (от 0 до 7) при выводе по адресу сегмента B800H
7-6 = 00 для всех текстовых режимов
= 01 для графических режимов с 16K
= 11 для графических режимов с 32K

4.5.4. Сдвиг между страницами текста

Поскольку страницы текста прилегают друг к другу в видеобуфере, небольшой текстовый массив может целиком помещаться в этой памяти. В этом случае текст может сдвигаться вверх и вниз по экрану, не передвигаясь реально в буфере. Вместо этого экран начинает показывать содержимое буфера, начиная с различных точек, и тем самым создается иллюзия сдвига. Этот метод называется аппаратным сдвигом.

Аппаратный сдвиг достигается за счет изменения стартового адреса вывода. Это число, указывающее на символ в видеобуфере, который будет выводиться в левом верхнем углу экрана. Добавление 80 к этому числу "сдвигает" весь экран на одну строку вверх, а вычитание 80 - на одну строку вниз. В режиме с 40 символами в строке надо вместо 80 прибавлять или вычитать 40. На рис. 4.7 приведена диаграмма аппаратного сдвига.

Отметим, что регистр стартового адреса не считает байты атрибутов, поэтому вы должны вычислять адреса памяти по-другому, а не так, как при прямом отображении в память. Имейте также в виду, что, несмотря на наличие разрывов памяти между границами страниц (96 байт между 80-символьными страницами и 48 байт между 40-символьными страницами), микросхема 6845 пропускает эти области и сдвиг непрерывно происходит с одной страницы на следующую. Аппаратный сдвиг происходит настолько

быстро, что вам может оказаться необходимым вставить процедуру задержки, чтобы пользователь имел возможность увидеть, насколько сдвинулся экран.

BIOS хранит текущее значение регистра стартового адреса в переменной в своей области данных. Эта двухбайтовая переменная расположена по адресу 0040:004EH.

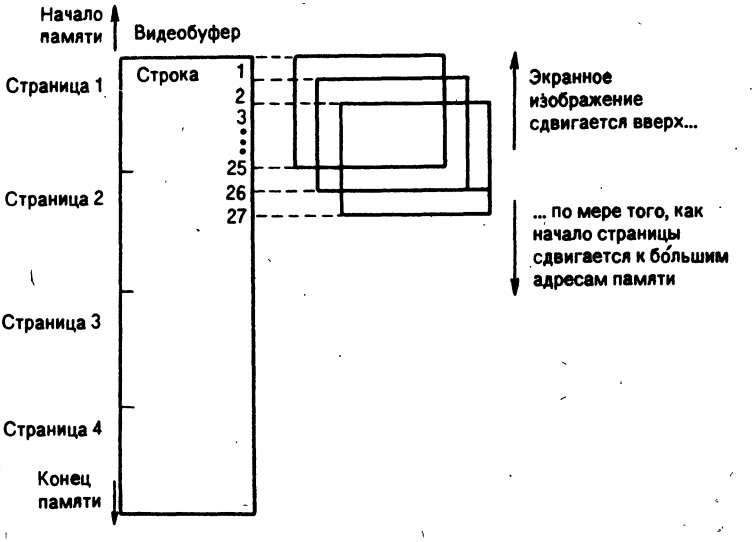


Рис. 4.7. Аппаратный сдвиг

Низкий уровень

Стартовый адрес содержится в регистрах 12 (старший байт) и 13 (младший байт) микросхемы 6845. В [4.1.1] объясняется работа этой микросхемы. Прежде чем адресуемый байт направляется в порт с адресом 3D5H, необходимо послать номер адресуемого регистра в порт 3D4H. В данном примере экран сдвигается вверх на одну строку. Переменная START_ADDR содержит адрес первого символа текущей верхней строки экрана.

```
MOV  BX,START_ADDR  ;начинаем с начала буфера
ADD  BX,80          ;сдвигаем на 1 строку (80 символов)
MOV  DX,3D4H        ;вывод в адресный регистр
MOV  AL,12          ;адресуем регистр 12
```

```
OUT   DX,AL          ;посылаем запрос
INC   DX             ;теперь выводим в командный регистр
MOV   AL,BH          ;старшее слово в AL
OUT   DX,AL          ;посылаем его в регистр 12
DEC   DX             ;обратно к адресному регистру
MOV   AL,13          ;адресуем регистр 13
OUT   DX,AL          ;посылаем запрос
INC   DX             ;снова командный регистр
MOV   AL,BL          ;младшее слово в AL
OUT   DX,AL          ;посылаем в регистр 13
```

Глава 5. Дисковые накопители

5.1. Управление распределением диска

Все диски как гибкие, так и жесткие организованы одинаковым образом. Поверхность диска разделена на ряд концентрических колец, называемых дорожками, а дорожки в свою очередь делятся радиально на сектора. Например, стандартная дискета с диаметром 5 1/4 дюйма имеет 40 дорожек, в системе MS-DOS 2.0 каждая такая дорожка разбита на 9 секторов (15 секторов на дискете емкостью 1.2 Мбайта и 17 - на фиксированном диске). Размер сектора - 512 байт, и 512 байт x 9 секторов x 40 дорожек x 2 стороны дает в итоге емкость дискеты 360K. Для всех типов дисков в MS-DOS размер сектора равен 512 байтам.

Файл располагают на таком количестве секторов, которое необходимо, чтобы вместить его полностью. Только несколько секторов на внешнем ободе дискеты зарезервированы для специальных нужд. Остальные доступны на основе правила "первый подошел - первого обслужат". Это означает, что по мере заполнения диска данными сектора постепенно заполняются по направлению к центру диска. При уничтожении файла сектора освобождаются и со временем свободные области становятся разбросанными по диску, разбивая новые файлы и замедляя доступ к ним для чтения и записи.

Фиксированные диски имеют некоторые специальные характеристики. Часто они состоят из двух или более параллельных пластин, у каждой из которых есть по две головки, чтобы читать с

обеих сторон. Все дорожки, расположенные на одинаковом расстоянии от центра, вместе называются цилиндром. Поскольку головки всех дисков двигаются тандемом, то можно достигнуть экономии перемещений, если заполнять все дорожки одного цилиндра, прежде чем переходить к следующему. Группы цилиндров могут относиться к различным операционным системам. Программа DOS FDISK может разбивать фиксированный диск на несколько разделов (до четырех) разного размера. По этой причине параметры фиксированного диска могут сильно отличаться.

Дисковые сектора определяются магнитной информацией, которую записывает утилита форматизации диска. Информация включает идентификационный номер каждого сектора. BIOS нумерует сектора 1-8, 1-9 или 1-15 в зависимости от емкости диска. Дорожки не маркируются, вместо этого они определяются механически по смещению головки чтения/записи от внешнего края диска. Дорожки нумеруются от 0 до 39 для дискет диаметром 5 1/4 дюйма, а для дисков большей емкости их может быть больше. Дисковые функции BIOS обращаются к определенному сектору, указывая номера дорожки и сектора. Однако функции DOS рассматривают все сектора диска как одну цепь, которая нумеруется подряд, начиная от 0, поэтому каждый сектор имеет свой логический номер.

На дискетах первый сектор (дорожка 0, сектор 1) содержит запись начальной загрузки, которая является небольшой программой, позволяющей компьютеру считывать с дискового накопителя остальные части MS-DOS. Затем следуют две копии таблицы размещения файлов, которые содержат информацию о распределении дискового пространства (вторая копия хранится из соображений безопасности). Затем идет корневой каталог, содержащий список файлов и ссылок на подкаталоги, а также указывающий, в каком месте диска они начинаются. Наконец, далее располагаются две небольшие программы DOS IBMBIO.COM и IBMDOS.COM. Они считываются при старте и обеспечивают компьютер возможностями, необходимыми для нахождения и загрузки файла COMMAND.COM, который, несомненно, является основной частью операционной системы.

Фиксированные диски имеют *главную запись загрузки*, включающую *таблицу разделов*, которая позволяет распределить диск между несколькими операционными системами. Таблица разделов несет в себе информацию о том, где на диске начинается раздел DOS, а также первый сектор какого раздела содержит запись начальной загрузки. В остальном раздел организован так же, как и дискета.

5.1.1. Чтение таблицы размещения файлов

Диск использует *таблицу размещения файлов* (FAT) для отведения дискового пространства файлам и хранения информации о свободных секторах. Из соображений безопасности на всех дисках хранятся по две копии FAT. Они хранятся последовательно, в секторах с самыми младшими доступными логическими номерами, начиная со стороны 0, дорожки 0, сектора 2 (сектор 1 занят записью начальной загрузки). Число секторов, занимаемых FAT, определяется размером и типом диска. Отметим, что в MS-DOS 3.0 размер записи FAT для фиксированного диска может быть равен 16 битам. Здесь мы будем рассматривать только 12-битовые записи; для получения информации о 16-битовых записях смотрите Техническое руководство по MS-DOS.

Таблица размещения файлов хранит информацию о каждом кластере секторов на диске. *Кластер* это группа стандартных секторов размером в 512 байт (независимо от типа диска MS-DOS всегда работает с 512-байтовыми секторами). Группировка секторов необходима, чтобы уменьшить размер FAT. Однако большие кластеры, используемые на фиксированном диске, напрасно расходуют дисковое пространство при записи маленьких файлов (утилита размером 500 байт занимает 4K дискового пространства). Ниже представлены размеры кластеров и размеры FAT для IBM PC:

Тип диска	Секторов на кластер	Размер FAT
дискета:		
160K	1	1
180K	1	1
320K	2	2
360K	2	2
1,2M	1	7
винчестер:		
10M	8	8
20M	4	40

При значительном размере кластера напрасно расходуется дисковое пространство, но когда большие диски имеют малый размер кластера, то таблица размещения файлов очень разрастается. При работе с дисками DOS загружает копию FAT в память, по возможности сохраняя ее там, поэтому при большом размере FAT может расходоваться много оперативной памяти.

Поскольку у машин AT достаточно памяти, то для них приемлемы намного бо́льшие FAT. Поэтому, обеспечивая экономию дискового пространства, для винчестера 20M взяты меньшие размеры кластеров, чем для 10M. Для дискет емкостью 1,2M выбран кластер размером в 1 сектор, так как их основное назначение состоит в хранении копий жесткого диска, а следовательно, компактность очень важна.

Каждая позиция в таблице размещения файлов соответствует определенной позиции кластера на диске. Обычно файл занимает несколько кластеров и запись в каталоге файлов содержит номер стартового кластера, в котором находится начало файла. Просмотрев позицию FAT, соответствующую первому кластеру, DOS находит номер кластера, в котором хранится следующая порция файла. Этому кластеру соответствует своя запись в FAT, которая в свою очередь содержит номер следующего кластера в цепочке. Для последнего кластера, занятого файлом, FAT содержит значения от FF8H до FFFH. Неиспользуемым (или освобожденным) кластерам соответствует значение 000, а дефектным секторам - FF7H. Наконец, значения от FF0H до FF7H приписываются резервным кластерам.

Номер кластера содержит 3 шестнадцатиричные цифры, для хранения которых требуется 1 1/2 байта. Для уменьшения размеров FAT числа для двух соседних кластеров хранятся в трех последовательных байтах таблицы. MS-DOS автоматически производит все необходимые вычисления.

Первые три байта FAT не используются для номеров кластеров. Первый байт содержит код, определяющий тип диска (см. [1.1.5]), а следующие 2 байта равны FFH. Поскольку эти позиции таблицы заняты, то кластеры нумеруются, начиная с 2, причем кластеры 2 и 3 занимают вторую тройку байтов таблицы.

MS-DOS 3.0 может создавать FAT с записями размером 16 бит. Такие записи необходимы для фиксированных дисков емкостью более 10M, которые имеют больше, чем 4086 кластеров. На рис. 5.1 показана связь между FAT и кластерами на диске.

Очень редко возникают причины вносить изменения прямо в таблицу размещения файлов. MS-DOS заботится обо всех файловых операциях и обеспечивает процедуры, анализирующие таблицу для определения свободного пространства на диске. Однако для некоторых специальных целей, таких, как восстановление удаленных файлов или написание драйвера блочного устройства, необходим прямой доступ к FAT. При прямом доступе к FAT надо соблюдать такие правила.

Для нахождения следующего кластера файла:

1. Умножьте номер кластера на 1,5.

2. Прочитайте 2 байта с полученным смещением (округляя вниз).
3. Если номер кластера четный, то возьмите младшие 12 бит, иначе возьмите старшие 12 бит.

Для преобразования номера кластера в логический номер сектора:

1. Вычтите 2 из номера кластера.
2. Умножьте результат на число секторов в кластере.

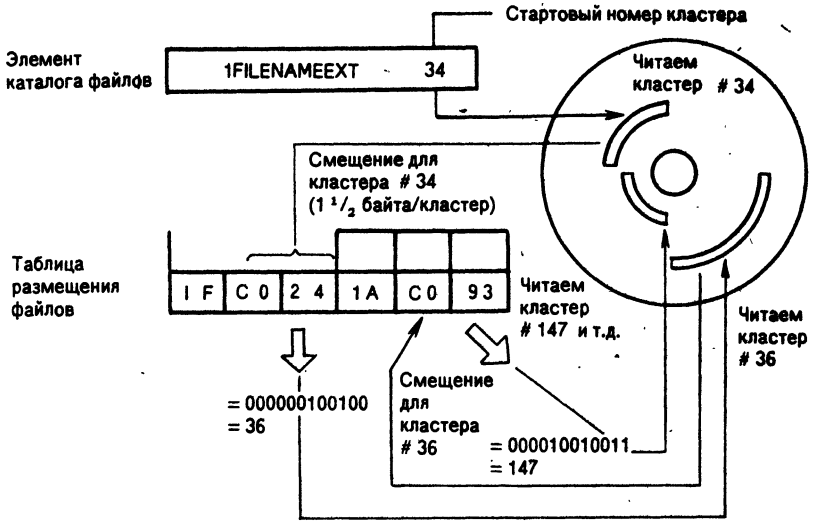


Рис. 5.1. Таблица размещения файлов

Высокий уровень

В следующем примере читается FAT и определяется значение, хранящееся для кластера с номером 6. В [5.4.2] объясняется начальный код, читающий сектора FAT. Результатом является 12-битовое число, представленное в виде трех шестнадцатиричных цифр (по 4 бита каждая), возвращаемое в виде строки. В примере пары чисел, состоящих из двух цифр, объединены, и в качестве результата берутся правые или левые три цифры. Когда Бейсик преобразует символ в шестнадцатиричную форму, то он возвращает только одну цифру, если первая была нулем, поэтому уда-

ленный нуль должен быть восстановлен, чтобы этот алгоритм работал правильно.

```
100 '''чтение секторов FAT
110 DEFINT A-Z
120 DATA &H55, &H8B, &HEC, &H1E, &H8B, &H76, &H0C, &H8B
130 DATA &H04, &H8B, &H76, &H0A, &H8B, &H14, &H8B, &H76
140 DATA &H08, &H8B, &H0C, &H8B, &H76, &H06, &H8A, &H1C
150 DATA &H8E, &HD8, &H8B, &HC3, &HBB, &H00, &H100, &HCD
160 DATA &H25, &H59, &H1F, &H5D, &HCA, &H08, &H100
170 DEF SEG = &H1000                    'помещаем машинный код с этого
'адреса
180 FOR N = 0 TO 38                     'читаем 39 байт данных
190 READ Q: POKE N,Q                    'переносим их в память
200 NEXT                                'следующий байт
210 READSEC = 0                         'начинаем процедуру с 1-го байта
220 BUFER = &H2000                      'адрес буфера приема данных
230 LOGNUM = 1                          'начальные сектора FAT
240 NUMSEC = 2                          '2 сектора в FAT
250 DRIVE = 0                           'читаем накопитель A
260 CALL READSEC(BUFER,LOGNUM,NUMSEC,DRIVE) 'читаем сектора
270 '''определяем номер следующего кластера файла
280 DEF SEG = &H2000                    'буфер, где хранится FAT
290 CLUSNUM! = 6                        'кластер номер 6
300 C! = CLUSNUM!                       'делаем копию
310 C! = INT (C!*1.5)                   'умножаем на 1,5 и округляем
320 X = PEEK(C!)                        'читаем 2 байта с этой позиции
330 Y = PEEK(C! + 1)
340 X$ = HEX$(X): Y$ = HEX$(Y)          'переводим в шестнадцати-
                                        'ричные строки
350 IF LEN(X$) = 1 THEN X$ = "0" + X$   'делаем их 2-символьными
360 IF LEN(Y$) = 1 THEN Y$ = "0" + Y$
370 H$ = Y$ + X$                        'объединяем числа в одну строку
380 '''проверяем кластер на четность
390 IF CLUSNUM! MOD 2 <> 0 THEN 420     'уход, если нечетный
400 NEXTCLUSTER$ = RIGHT$(H$,3)         'если четный, то правые 3 цифры
410 GOTO 430
420 NEXTCLUSTER$ = LEFT$(H$,3)          'если нечетный, то левые
430 PRINT NEXTCLUSTER$                  'печатаем результат
```

Средний уровень

Функция DOS 1CH предоставляет информацию о таблице размещения файлов, но не саму FAT. Поместите номер накопителя в DL, где 0 = накопитель по умолчанию, 1 = A и т.д. При

возврате DX содержит число кластеров в FAT, а CX - число байтов в секторе. DS:BX указывают на байт, содержащий первый байт FAT, т.е. на код, указывающий тип диска; эти коды перечислены в [1.1.5].

Низкий уровень

Намного легче получить доступ к FAT в языке ассемблера. Отметим, что умножение номера кластера на 1,5 производится копированием числа, сдвигом копии вправо на 1 бит для деления пополам и сложением копии с оригиналом. Этот метод автоматически округляет результат вниз. Код, считывающий сектора FAT в память, обсуждается в [5.4.2].

```
;---в сегменте данных
BUF     DB    1024 DUP(0)           ;отводим место для 2 секторов
;---читаем FAT в память
        LEA   BX,BUF                ;указываем на буфер данных
        MOV   DX,1                  ;логический номер сектора
        MOV   CX,2                  ;2 сектора
        MOV   AL,0                  ;накопитель A
        INT   25H                   ;читаем сектора
        POP   CX                    ;восстанавливаем стек
;---получаем номер кластера
        MOV   AX,3                  ;номер кластера в AX
        MOV   CX,AX                 ;делаем копию
        MOV   DX,AX                 ;делаем вторую копию
        SHR   DX,1                  ;делим вторую копию на 2
        ADD   CX,DX                 ;складываем между собой
        ADD   BX,CX                 ;добавляем как смещение
        MOV   DX,[BX]               ;получаем 2 байта из этого места
        TEST  AX,1                  ;номер кластера нечетный?
        JNZ   ODD_CL                ;уход, если да
        AND   DX,0000111111111111B  ;получаем номер
        JMP   SHORT CONT            ;уход через обработку нечетного
ODD_CL: MOV   CL,4                  ;подготовка к сдвигу вправо
        SHR   DX,CL                 ;сдвигаем вниз старшие 12 бит
CONT:                               ;теперь следующий кластер в DX
```

5.1.2. Определение доступного дискового пространства

Хотя в следующем параграфе объяснено, как восстановить ситуацию при ошибке из-за нехватки места на диске, но нет лучшего "лекарства", чем предусмотрительность. Программа

должна контролировать доступное дисковое пространство и сообщать пользователю о нехватке места. Если места недостаточно, то пользователь может выйти из программы и устранить проблему без потери информации.

Высокий уровень

Следующая ассемблерная подпрограмма возвращает в переменную CLUSTERS число свободных кластеров на указанном диске. Надо поместить номер накопителя в DRIVENUM, где 1 = A, 2 = B и т.д. В приложении Г объясняется, как ассемблерные подпрограммы включаются в программы на Бейсике.

```
10 DEFINT A-Z                          'используем целые переменные
20 DRIVENUM = 1                        'сюда помещаем номер накопителя
30 CLUST = 0                           'инициализируем переменную
40 DATA &H55, &H8B, &HEC, &H8B, &H76, &H06, &H8B
50 DATA &H14, &HB4, &H36, &HCD, &H21, &H8B, &H7E
60 DATA &H08, &H89, &H1D, &H5D, &HCA, &H04, &H00
70 DEF SEG = &H1000                    'помещаем подпрограмму
80 FOR N = 0 TO 20                     'берем каждый байт
90 READ Q: POKE N,Q                    'читаем его и помещаем в память
100 NEXT
110 FREESP = 0                         'указатель на начало процедуры
120 CALL FREESP(CLUST,DRIVENUM)        'вызов процедуры
130 PRINT "CLUSTERS: ";CLUST           'печать числа кластеров
```

Средний уровень

Функция 36H прерывания 21H сообщает, сколько имеется свободного пространства на диске. Единственный входной регистр - DL, который должен содержать номер накопителя. Накопитель по умолчанию обозначается 0, накопитель A - 1 и т.д. При возврате BX содержит число доступных кластеров, AX - число секторов в кластере, а CX - количество байтов в секторе. Небольшое упражнение в умножении дает желаемый результат. В следующем примере проверяется, что на двухсторонней дискете осталось по меньшей мере 2K дискового пространства:

```
MOV  AH,36H        ;номер функции
MOV  DL,1          ;накопитель A
INT  21H           ;получаем информацию
```

```
CMP   BX,2            ;имеется ли 2 свободных кластера?
JL    RUNNING_OUT     ;если нет, то сообщаем об этом
```

5.1.3. Получение/установка размера файла

Программе может понадобиться проверить размер файла по разным причинам. Одна из возможных причин состоит в определении числа записей, содержащихся в файле. Другая - в определении позиции конца файла, с тем чтобы файловый указатель был установлен верно для добавления в файл новых данных без изменения существующих.

Конечно, размер файла устанавливается автоматически функцией DOS. Иногда программа может нуждаться в резервировании дискового пространства для дальнейшего использования. В этом случае надо открыть файл в режиме прямого доступа и вывести такой номер записи, чтобы файл имел достаточную длину. Записи между "фиктивной" и реально относящимися к файлу будут заполнены теми данными, которые случайно окажутся в дисковых секторах, отведенных для файла при этой операции.

Высокий уровень

В Бейсике функция LOF (длина файла) возвращает точное число байтов, отведенных файлу (предупреждаем, однако, что старые версии Бейсика - 1.x - возвращают число 128-байтовых блоков, используемых файлом). Файл должен быть уже открыт и ссылаться на него надо по номеру, под которым он был открыт. Формат X = LOF(1). В следующем примере определяется, сколько 64-байтовых записей содержится в файле, открытом как #3:

```
100 OPEN "FILENAME" AS #3              'открываем файл
110 RECORDLEN = 64                     'определяем длину записи
120 NUMBREC = LOF(3)/RECORDLEN         'вычисляем число записей
```

Средний уровень

FCB функция 23H прерывания 21H сообщает число записей в файле. Если приписать файлу длину записи в 1 байт, то его размер будет возвращен в байтах. DS:DX должны указывать на управляющий блок открытого файла. Затем вызовите функцию. Если файл не найден, то в AL возвращается FF. В противном случае в AL возвращается 0, а число записей помещается в поле номера записи прямого доступа FCB (байты 33-36). Для правиль-

ной работы поле длины записи FCB должно быть установлено после открытия файла, но перед вызовом функции; это двухбайтовое поле расположено по смещению 14 в FCB. Если размер файла неточно делится на длину записи, то сообщаемое число записей округляется вверх. Вот пример, в котором используется длина записи, равная 1:

```
;---определение размера файла
LEA   DX,FCB          ;DS:DX указывает на FCB
MOV   BX,DX           ;копируем указатель в BX
MOV   CX,1            ;размер записи в CX
MOV   [BX] + 14,CX    ;пишем в поле размера записи FCB
MOV   AH,23H          ;функция, сообщающая размер файла
INT   21H             ;вызов функции
MOV   AX,[BX] + 33    ;получаем младшую часть размера файла
MOV   CX,[BX] + 35    ;получаем старшую часть размера файла
```

Можно также устанавливать длину файла, применяя управляющий блок файла. Для этого надо использовать функцию записи блока с прямым доступом, которая обсуждается в [5.4.5]. У этой функции имеется частный случай, когда число выводимых записей устанавливается равным нулю, то длина файла устанавливается равной числу записей, указанному в поле записи прямого доступа.

Метод, использующий дескриптор файла (file handle) не имеет функции, которая непосредственно сообщала бы длину файла, однако имеется возможность вычислить размер, передвинув указатель файла с его начала на конец. При открытии файла указатель файла автоматически устанавливается на первый его байт. Указатель файла перемещается функцией 42H прерывания 21H. Надо поместить в AL кодовое число 2, направляющее указатель на конец файла. В BX должен быть указан номер файла, а CX:DX содержит смещение от конца файла до позиции, в которую должен быть установлен указатель, поэтому поместите 0 в оба этих регистра. Затем вызовите функцию. При возврате DX:AX будет содержать новую позицию указателя относительно его предыдущей позиции, т.е. будет содержать длину файла (DX содержит старший байт). При возникновении ошибки будет установлен флаг переноса, а в AX будет возвращена 1, если неправилен номер функции, и 6, если неправилен номер файла. Не забудьте затем снова вернуть указатель на начало файла, если это необходимо. Поместите 0 в AL, CX и DX и вызовите функцию снова. Продемонстрируем сказанное на примере:

```
;---открываем файл
LEA   DX,FILE_PATH     ;DS:DX указывают на путь к файлу
MOV   AL,0             ;открываем файл для чтения
MOV   AH,3DH           ;функция открытия файла
INT   21H              ;открываем его
JC    OPEN_ERROR       ;проверка на ошибку
MOV   HANDLE,AX        ;запоминаем номер файла
;---определяем длину файла
MOV   AH,42H           ;функция перемещения указателя
MOV   AL,2             ;код установки на конец файла
MOV   BX,HANDLE        ;номер файла в BX
MOV   CX,0             ;0 в CX и DX
MOV   DX,0
INT   21H              ;сдвигаем указатель
JC    POINTER_ERROR    ;ошибка?
MOV   FSIZE_HIGH,DX    ;запоминаем размер файла
MOV   FSIZE_LOW,DX
```

5.1.4. Восстановление после ошибок, связанных с нехваткой пространства на диске

При попытке записи на полный диск может произойти сбой программы. Часто легко избежать этого, даже в Бейсике, проверив предварительно наличие дискового пространства (см. [5.1.2]). Однако если ошибка все же произошла, то постарайтесь дать пользователю возможность исправить ее. Позвольте ему сохранить только часть данных или стереть какой-нибудь другой файл и повторить попытку. Или, еще более радикальное средство, позвольте пользователю вставить другую дискету. Последний подход должен реализовываться с большой осторожностью. Сначала закройте все открытые файлы. Затем выдайте запрос на смену дискеты. После того как пользователь сообщит, что новая дискета вставлена, создайте новый файл и запишите туда данные.

Высокий уровень

В Бейсике надо установить процедуру обработки ошибок, как показано в [7.2.5]. Если оператор Бейсика делает попытку писать на полный диск, то возвращается код ошибки #61. При этом управление может быть передано процедуре обработки ошибок, которая информирует пользователя о проблеме и позволяет ему справиться с ней, не теряя данных.

```
100 ON ERROR GOTO 5000                    'разрешаем обработку ошибок
.
.
200 OPEN FNAME$ FOR OUTPUT AS #1          'открываем файл
210 FOR N = 1 TO ARRLEN                   'начинаем писать массив на диск
220 PRINT #1, ARRAY$(N)                   'записываем один элемент
230 NEXT                                  'следующий
.
.
5000 IF ERR = 61 THEN 6100                'диск полон?
5100 IF ERR = ...                         'другие ошибки ...
.
6100 '''восстановление при переполнении диска
6110 BEEP: PRINT "Disk full - choose an option:"
6120 PRINT "(A) - Re-edit the file"
6130 PRINT "(B) - Delete some other file from disk"
6140 PRINT "(C) - Use different diskette"
.
(здесь идет процедура восстановления)
.
6500 RESUME
```

Средний уровень

Все функции DOS, которые пишут на диск, выдают определенный код ошибки при попытке записи на полный диск. Приведем сводку этих кодов:

Метод доступа	Функция	Название	Код ошибки
FCB	15H	Последовательная запись	AL = 1
FCB	22H	Прямая запись	AL = 1
FCB	27H	Прямая запись блока	AL = 1
Дескриптор	40H	Запись в файл/устройство	CX <> BX

Проверка на ошибку должна делаться после каждой записи на диск. И если критической ошибки не происходит, то восстановление не вызывает проблем. Надо только проверять на ошибку каждый раз, когда вы вызываете одну из этих функций, и создать надежную процедуру обработки ошибок по вашему вкусу.

5.2. Работа с каталогами диска

Каждый диск имеет один корневой каталог, с которого начинается поиск всех остальных каталогов. Корневой каталог может содержать элементы, указывающие на подкаталоги, которые в свою очередь могут содержать ссылки на другие подкаталоги, образуя древовидную структуру каталогов. Корневой каталог всегда расположен в определенных секторах диска; подкаталоги хранятся как обычные дисковые файлы, поэтому они могут быть расположены в любом месте диска. Отметим, что если фиксированный диск разбит на разделы, он может содержать до четырех корневых каталогов, хотя MS-DOS "видит" только один корневой каталог. Каталоги имеют различные размеры в зависимости от размера диска и его разбиения на разделы. В следующей таблице приведены размеры и позиции корневых каталогов для разных типов дисков:

Тип диска		Размер каталога	Число элементов	Начальный сектор
дискета	160K	4 сектора	64	9
	180K	4 сектора	64	9
	320K	7 секторов	112	15
	360K	7 секторов	112	15
	1,2M	14 секторов	224	29
жесткий диск	10M	----------переменные---------		
	20M	----------переменные---------		

В зависимости от разбиения на разделы фиксированный диск может иметь различные размеры каталога и номер начального сектора. Если весь диск отведен для MS-DOS, то на XT и AT под корневой каталог отводится 32 сектора, что позволяет иметь в нем 512 элементов.

Как корневой каталог, так и подкаталоги используют 32 байта для хранения информации об одном файле, независимо от типа диска. Таким образом, в каждом секторе может храниться информация о 16 элементах каталога. Каждое 32-байтовое поле разбито следующим образом:

байты	0-7	имя файла
	8-10	расширение файла
	11	атрибут файла
	12-21	зарезервировано
	22-23	время последнего доступа к файлу
	24-25	дата последнего доступа к файлу
	26-27	начальный кластер
	28-31	размер файла

Точка между именем файла и его трехбайтовым расширением не хранится. Все поля выравнены по левой границе, а пустые байты заполняются пробелами (код ASCII 32). Атрибут файла определяет, является ли файл спрятанным, защищенным от записи и т.д. (см. [5.2.6]). Он определяет также специальные элементы каталога, такие, как подкаталоги или метка тома. Информация о времени и дате упакована, поэтому для чтения этих значений требуются битовые операции (см. [5.2.5]).

Начальный кластер указывает на позицию в таблице размещения файлов (FAT), которая обсуждалась в [5.1.1]. FAT хранит информацию о свободном пространстве на диске, а также отводит сектора при записи файла. FAT отводит дисковое пространство порциями, большими чем 1 сектор, которые называются кластерами. Файл расположен в цепочке кластеров, и FAT содержит соответствующую цепочку элементов, указывающих, где эти кластеры расположены на диске. Каталог должен указывать на начальное звено цепочки элементов файла в FAT, и эта информация содержится в поле *начальный номер кластера*. Поскольку файл обычно не целиком занимает последний отведенный ему кластер, то поле *размер файла* хранит точную длину файла в байтах.

5.2.1. Чтение/изменение корневого каталога

Каталоги диска подразделяются на корневой каталог (обсуждаемый здесь) и подкаталоги (обсуждаемые в [5.2.3]). Когда пользователь программы вводит для работы имя какого-либо файла, то бывает полезным проверить, имеется ли этот файл на самом деле. Обычно изменения в корневом каталоге производятся в ходе обычных файловых операций или с помощью специальных функций DOS. Однако можно работать с каталогом напрямую. Необходимость в таком подходе возникает при работе на языках высокого уровня, где утилиты DOS по большей части недоступны.

Корневой каталог читается и изменяется при загрузке его в память с помощью метода, показанного в [5.4.2], когда читаются

абсолютные сектора диска. В этом случае не остается места между секторами, когда они загружаются в память. Буфер, содержащий данные сектора, может рассматриваться как набор 32-байтовых полей и пара указателей, которые могут использоваться для движения по каталогу. Один указатель всегда кратен 32 и указывает на начало элемента каталога. Второй указатель добавляется к первому и указывает на одно из полей в 32-байтовом элементе. Данные в памяти могут быть изменены требуемым образом, а затем весь буфер записывается обратно на диск.

Имеются два метода чтения абсолютных секторов диска, и в обоих случаях только одно число отличает чтение от записи. Поскольку ошибка при записи на диск может легко повредить все содержимое диска, то надо действовать аккуратно. Сначала убедитесь, что операция чтения сектора выполнена верно во всех отношениях. После этого можно попробовать записать на диск, взяв точную копию кода, использованного для чтения, и заменив только номер функции.

Высокий уровень

Бейсик выводит каталог по команде FILES. При этом выводятся только имена файлов. FILES выдает каталог накопителя по умолчанию; для указания накопителя напишите FILES "A:" и т.д. Можно потребовать, чтобы была выведена информация об отдельном файле, написав FILES "A:MYFILE.DAT". Как и в операционной системе, имя файла может содержать * и ?. Оператор FILES снабжает информацией пользователя, но иногда наличие некоторого файла "хочет" проверить программа. В этом случае надо открыть файл для последовательного чтения и, если он не существует, возникнет ошибочная ситуация. Смотрите обсуждение и пример в [5.2.3].

Для поиска любой информации, относящейся к корневому каталогу, используйте процедуру на машинном языке, приведенную в [5.4.2]. После того как данные каталога помещены в память, установите указатели, как описано выше, и ведите поиск по буферу памяти с 32-байтовым интервалом. В приведенном ниже примере показан поиск элемента каталога, относящегося к стертому файлу. Когда файл стирается, первый байт имени файла заменяется на E5H, но все остальное содержимое данного элемента остается неизменным. Конечно, при этом освобождается дисковое пространство, отведенное файлу в FAT. Процедура восстановления удаленного файла должна знать номер начального кластера в FAT. В примере этот двухбайтовый номер кластера помещается со смещением 26 в элементе каталога.

```
100 '''чтение секторов каталога в память с сегмента &H2000
110 INPUT "Enter erased filename ", FNAM$        'запрос имени файла
120 IF LEN(FNAM$) > 12 THEN BEEP: GOTO 110      'ошибка
130 IF INSTR(FNAM$,".") > 9 THEN BEEP: GOTO 110 'ошибка
140 '''дополнение имени и расширения файла нулями
150 Y = INSTR(FNAM$,".")                        'проверяем положение точки
160 IF Y = 0 THEN FIRPAR$ = FNAM$: GOTO 230 'если типа нет, то вперед
170 EXT$ = LEFT$(FNAM$, LEN(FNAM$) - Y) 'выделяем тип
180 EXT$ = EXT$ + STRING$(3 - LEN(EXT$),"")     'добавляем пробелы
190 FIRPAR$ = RIGHT$(FNAM$,Y - 1)
200 FIRPAR$ = FIRPAR$ + STRING$(8 - LEN(FIRPAR$),"")
210 FNAM$ = FIRPAR$ + EXT$
220 '''теперь хотим найти удаленный файл
230 MID$(FNAM$,1,1) = CHR$(&HE5)          'заменяем первый символ
240 DIRPTR = 0                            'указатель на элемент
250 FIELDPTR = 26                         'указатель на номер кластера
260 FOR N = 1 TO 112                      'на дискете 112 элементов
270 X$ = ""                               'чистим X$
280 FOR M = 0 TO 10                       'читаем имя файла из каталога
290 X$ = X$ + PEEK(DIRPTR + M)            'берем по символу
300 NEXT                                  'следующий
310 IF X$ = FNAM$ THEN 340                'совпадает с введенной строкой
320 NEXT                                  'если нет, то следующий
330 PRINT "Too late - file entry obliterated": END      'не найден
340 X = PEEK(DIRPTR + FIELDPTR)           'нашли его, берем 1-й байт и
350 Y = PEEK(DIRPTR + FIELDPTR + 1)       '2-й байт номера кластера
360 Z = X + 256*Y                         'теперь номер кластера в Z
```

Средний уровень

MS-DOS обеспечивает две пары функций для поиска файлов: одна - для файлов, открытых методом управляющих блоков файла, а другая - для файлов, открытых методом дескриптора файлов. Одна из функций каждой пары ищет первое появление имени файла в каталоге, а другая - последующие появления, когда в имени файла содержатся "джокеры"*. Только метод, использующий дескриптор файла, позволяет искать подкаталоги.

Метод FCB. Функция 11H прерывания 21H ищет первое появление файла. Установите DS:DX на неоткрытый FCB и выполните

*Джокер - это карта, которая может заменить любую другую по желанию ее обладателя. В вычислительной технике имеются два вида "джокеров": ?, заменяющий ровно один любой символ, и *, заменяющий любое количество любых символов. - *Примеч. пер.*

функцию. При возврате AL будет содержать 0, если файл найден, и FF - в противном случае. DTA заполняется информацией из каталога. Для обычных FCB первый байт DTA содержит номер накопителя (1 = A и т.д.), а следующие 32 байта содержат элемент каталога. Для расширенного FCB первые 7 байт файла копируются в первые 7 байт расширенного FCB, восьмой байт указывает на накопитель, а следующие 32 байта - элемент каталога.

```
;---в сегменте данных
FCB DB   1,'NEWDATABAK',25DUP(0)
;---ищем файл
     MOV AH,11H    ;функция поиска в каталоге
     LEA DX,FCB    ;указываем на FCB
     INT 21H       ;ищем
     CMP AL,0      ;успешно?
     JNE NO_FILE   ;если нет, то действует процедура обработки ошибки
     LEA BX,DTA    ;теперь DS:BX указывает на элемент каталога
```

После функции 11H можно использовать функцию 12H для поиска следующих подходящих элементов, когда имя файла содержит "джокеры". В данном случае в имени файла допустим только символ "?", но не "*". Эта функция работает точно так же, как и первая, и если найден второй файл, то информация о первом файле в DTA будет уничтожена повторной записью.

Метод дескриптора файлов. Функция 4EH прерывания 21H ищет файл с данным именем. DS:DX должны указывать на строку, представляющую путь к файлу. Например, B:\EUROPE\FRANCE\PARIS указывает на файл PARIS. Строка может содержать до 63 символов и завершаться символом ASCII 0. Имя файла может содержать "джокеры", включая как "?", так и "*". Поместите атрибут файла в CX; если он обычный, то 0, в противном случае проконсультируйтесь в [5.2.6] относительно значений атрибута.

При возврате устанавливается флаг переноса, если файл не найден. Если файл найден, функция заполняет DTA информацией о нем. Ниже приведен частный случай применения DTA с методом дескриптора файлов, хотя обычно DTA используется функциями MS-DOS для работы с помощью FCB. Первые 21 байт DTA зарезервированы DOS для поиска следующих совпадающих файлов. Двадцать второй байт соответствует атрибуту файла, за ним идут два байта, содержащие время, и еще два байта - дату. Следующие 4 байта содержат размер файла, начиная с младшего слова. И наконец, дается имя файла в виде строки переменной длины,

4 байта содержат размер файла, начиная с младшего слова. И наконец, дается имя файла в виде строки переменной длины, заканчивающейся байтом ASCII 0. Точка (ASCII 46) разделяет имя и расширение, и ни один из этих элементов не заполнен пробелами.

```
;---в сегменте данных
PATH DB   'B:FRANCE\PARIS\4EME',0
;---ищем файл
      MOV  AH,4EH          ;номер функции
      LEA  DX,PATH         ;DS:DX указывают на путь
      MOV  CX,0            ;обычный атрибут файла
      INT  21H             ;ищем файл
      JC   NO_FILE         ;уход, если не найден
      LEA  BX,DTA          ;DS:BX указывают на DTA
      MOV  AL,[BX] + 21    ;теперь атрибут файла в AL
```

Поиск следующего появления имени файла (когда используются "джокеры") проводится с помощью функции 4FH прерывания 21H. Она подготавливается так же, как и функция 4EH, при этом указатель DTA не должен меняться. Если других совпадений не найдено, устанавливается флаг переноса, а в AX появляется 18.

5.2.2. Создание/удаление подкаталога

Программа может создавать или удалять подкаталоги только при выполнении некоторых условий. Для создания подкаталога необходимо, чтобы было по крайней мере одно пустое место в корневом каталоге, а для удаления подкаталога - чтобы он не содержал файлов или ссылок на другие подкаталоги. Кроме того, вы не можете удалить подкаталог, который является вашим текущим каталогом (т.е. тот, с которым по умолчанию выполняются все операции над каталогами). Отметим также, что невозможно удалить корневой каталог.

Высокий уровень

Бейсик предоставляет команды MKDIR (создать каталог) и RMDIR (удалить каталог), после чего должны следовать стандартные пути указания каталога, содержащие до 63 символов, включая имя накопителя. Путь должен быть заключен в кавычки. Что-

бы добавить подкаталог с именем STORKS в подкаталог BIRDS, напишите MKDIR "B:MAMMALS\BIRDS\STORKS". После выполнения этой команды будет создан файл STORKS, используемый как подкаталог, и факт его существования будет отражен в создании элемента с именем STORKS в подкаталоге с именем BIRDS. Для удаления этого подкаталога надо сначала удалить из него все файлы (см. [5.3.2]). Затем надо использовать команду RMDIR "B:MAMMALS\BIRDS\STORKS".

В этих примерах предполагалось, что вашим текущим каталогом являлся корневой каталог. Однако если он находится где-то на пути к подкаталогу, над которым осуществляются операции, то нет необходимости указывать весь путь. Поэтому если вашим текущим каталогом является BIRDS, то для создания или удаления подкаталога STORKS можно использовать команды MKDIR "\STORKS" или RMDIR "\STORKS".

Средний уровень

Поскольку управляющие блоки файлов обслуживают только корневой каталог, для создания или удаления подкаталога надо использовать дескрипторы файлов.

Создание подкаталога. DS:DX должны указывать на строку, дающую накопитель и путь к каталогу, в котором необходимо создать подкаталог. Строка должна завершаться байтом ASCII 0. Для открытия подкаталога с именем PRIMATES в корневом каталоге накопителя A: записывают строку в виде "A:\PRIMATES". Для открытия подкаталога в другом подкаталоге с именем MAMMALS напишите "A:\MAMMALS\PRIMATES". Имя накопителя A: может быть опущено, если вы работаете с накопителем, используемым по умолчанию, и путь может начинаться с текущего каталога. Поместите в AH 39H и выполните прерывание 21H; если указан правильный путь, то будет создан новый каталог. В противном случае будет установлен флаг переноса, а AX - содержать код ошибки 3 (путь неверен) или 5 (нет доступа). В приведенном примере создается подкаталог PRIMATES:

```
;---в сегменте данных
PATH DB   'A:MAMMALS\PRIMATES',0
;---создаем подкаталог с именем PRIMATES
      LEA  DX,PATH        ;DS:DX должны указывать на путь
      MOV  AH,39H         ;номер функции
      INT  21H            ;создаем подкаталог
      JC   ERROR_ROUT     ;обработка ошибок
```

Удаление подкаталога. Для удаления подкаталога надо сформировать строку, полностью совпадающую с той, которую вы указывали при создании каталога. Затем поместите в AH 3AH и выполните прерывание 21H. Опять при невыполнении функции в AX будут возвращены коды 3 или 5 (код 5 указывает на то, что каталог непустой).

5.2.3. Чтение/изменение подкаталога

Подкаталоги во многом подобны корневому каталогу, за исключением того, что они хранятся как обычные файлы, а не в заранее определенных секторах. Подкаталоги невозможно спутать с обычными файлами, поскольку объект каталога, относящийся к подкаталогу, имеет специальный байт атрибутов (с установленным битом 5, см. [5.2.6]). Подкаталоги начинаются с двух специальных 32-байтовых объектов, первый из которых имеет имя "точка", а второй - "две точки". Они ориентируют подкаталог среди окружающих каталогов. Ссылки на подкаталоги нижнего уровня записываются как обычные ссылки на файлы.

Предполагается, что подкаталог может быть прочитан как любой другой файл, поэтому, казалось бы, не составляет труда загрузить его в память. Но, к сожалению, создатели MS-DOS поместили 0 в поле длины файла для элементов, относящихся к подкаталогам. В результате DOS считает, что этот файл имеет нулевую длину, и отказывается читать его. Нет простого способа преодолеть эту проблему.

Высокий уровень

В Бейсике команда FILES может использовать стандартные имена путей для вывода подкаталога; например, FILES "B:MAMMALS\BIRDS" выводит все файлы, содержащиеся в подкаталоге BIRDS. Эта команда может быть применена и для получения информации о наличии в каталоге определенного файла. Например, FILES "LEVEL1\NEWDATA" ищет файл NEWDATA и выводит его имя, если он найден. Хотя это может быть полезным для пользователя, но часто самой программе необходимо знать, существует или нет указанный файл. Чтобы установить это, попытайтесь открыть файл для последовательного чтения. Если он не будет найден, то возникнет ошибочное условие 63. Создайте процедуру обработки ошибок, как описано в [5.4.8]. Затем используйте переменную, чтобы отметить, был ли найден требуемый файл (в нашем примере переменная "EXISTS"). Если

нет необходимости, чтобы этот файл оставался открытым, то закройте его перед тем, как двинуться дальше.

```
100 ON ERROR GOTO 1000          'процедура обработки ошибок
110 EXISTS = 1                  'начальное значение "флага"
120 INPUT "Enter filename: ",S$  'запрос имени файла
130 OPEN S$ FOR INPUT AS #3     'открываем его для последующего чтения
140 IF EXISTS = 0 THEN BEEP: PRINT "File does not exist"
.
.
1000 IF ERR = 53 THEN 1500      'файл не существует?
1010 IF ERR = 64 THEN ...       'другие ошибки
.
1500 EXISTS = 0                 'меняем значение флага
1510 RESUME 140                 'продолжаем выполнение программы
```

Средний уровень

Функции, применяющие метод дескриптора файлов, которые использовались для доступа к корневому каталогу (см. [5.2.1]), могут так же просто обращаться к любому подкаталогу. Чтобы вывести все содержимое каталога, надо использовать функцию 4EH для поиска файлов *.*, а затем повторять поиск, применяя функцию 4FH. Когда искомых файлов больше не останется, то устанавливается флаг переноса, а AL будет содержать 18. Каждый раз при обнаружении очередного элемента в DTA будет записываться информация о файле, включая полный путь к нему (отмечаем использование DTA в функциях, работающих с дескриптором файла). В следующем примере выводятся полные пути ко всем обычным файлам подкаталога.

```
;---в сегменте данных
PATH    DB   'A:MAMMALS\*.*',0
DTAH    DB   256 DUP(?)
;---установка DTA
        LEA  DX,DTA          ;DS:DX указывают на DTA
        MOV AH,1AH           ;функция установки DTA
        INT  21H             ;устанавливаем DTA
;---ищем первый файл
        MOV AH,4EH           ;номер функции
        LEA  DX,PATH         ;указываем на строку пути
```

```
            MOV CX,0                ;только нормальные атрибуты
            INT 21H                 ;ищем *.*
            JC  ERROR               ;обработка ошибок
;---выводим имя файла
NEXT_LI:    LEA BX,DTA              ;BX указывает на DTA
            ADD BX,30               ;смещение для имени файла
NEXT_CH:    MOV DL,[BX]             ;получаем символ из имени
            CMP DL,0                ;проверка на конец строки
            JE  END_STR             ;выход, если конец
            MOV AH,2                ;иначе - выводим символ
            INT 21H
            INC BX                  ;увеличиваем указатель
            JMP SHORT NEXT_CH       ;следующий символ
;---возврат каретки/перевод строки в конце строки
END_STR:    MOV AH,2                ;функция вывода символа
            MOV DL,13               ;код возврата каретки
            INT 21H                 ;выводим
            MOV DL,10               ;код перевода строки
            INT 21H                 ;выводим
;---ищем следующий файл
            LEA DX,PATH             ;указываем на строку пути
            MOV AH,4FH              ;номер функции
            INT 21H                 ;ищем следующий файл
            JC  FINISH              ;если нет, то выход
            JMP SHORT NEXT_LI       ;иначе - выводим имя файла
FINISH:
```

5.2.4. Получение/установка текущего каталога

Текущий каталог это каталог, в котором DOS ищет файл, если к нему не указан путь. Если не установлено противного, то текущим каталогом является корневой каталог.

Высокий уровень

Бейсик устанавливает текущий каталог с помощью команды CHDIR. За командой следует строка, указывающая путь к каталогу, на который надо перейти. Строка может содержать до 63 символов, включая имя накопителя, и должна быть заключена в кавычки. CHDIR "C:MAMMALS\PRIMATES\GIBBONS" делает

подкаталог GIBBONS текущим каталогом. Чтобы перейти в корневой каталог, напишите CHDIR "\" или CHDIR "B:\".

Бейсик версии 3.0 может сообщать путь к текущему каталогу, как это делает команда DOS PATH. Введите PRINT ENVIRON$("PATH").

Средний уровень

Функция 3BH прерывания 21H устанавливает текущий каталог. DS:DX должны указывать на путь к каталогу в стандартном виде, и эта строка должна завершаться байтом ASCII 0. Например, B:BIRDS\PARROTS\POLLY делает POLLY текущим каталогом. B: может быть опущено, если это текущий накопитель по умолчанию (см. [5.3.1]). Чтобы сделать текущим корневой каталог накопителя A:, напишите A:\. В предлагаемом примере текущим каталогом устанавливается POLLY:

```
;---в сегменте данных
PATH  DB   'B:BIRDS\PARROTS\POLLY',0
;---делаем POLLY текущим каталогом
        MOV AH,3BH      ;номер функции
        LEA  DX,PATH    ;DS:DX должны указывать на путь
        INT  21H        ;устанавливаем текущий каталог
```

Чтобы определить, какой каталог является текущим, надо использовать функцию 47H прерывания 21H. DS:SI должны указывать на область данных размером 64 байта, в которую будет записан путь. В DL указывается накопитель, причем 0 = "по умолчанию", 1 = A, 2 = B и т.д. Функция возвращает строку без имени накопителя. Если был указан несуществующий накопитель, то в AL возвращается код ошибки 15. Строка начинается с имени первого подкаталога цепочки, а не с обратной косой черты. Байт ASCII 0 сигнализирует о конце строки. В данном примере имя текущего каталога присваивается переменной "CURRENT_DIR":

```
;---в сегменте данных
CUR_DIR DB   64 DUP(?)
;---получить текущий каталог
        MOV AH,47H              ;номер функции
        LEA  SI,CURRENT_DIR     ;указываем на область данных
        MOV DL,1                ;накопитель A
        INT  21H                ;помещает строку по адресу DS:SI
```

5.2.5. Получение/установка времени и даты последнего доступа к файлу

Если отсчитывать от нуля, то байты 22-23 32-байтового элемента каталога содержат время последнего доступа к файлу. Байты 24-25 содержат дату. Значение битов следующее:

Время:	биты	11-15	часы (0-23)
		5-10	минуты (0-59)
		0-4	секунды (0-29 с двухсекундным интервалом)
Дата:	биты	9-15	год (0-119, смещение с 1980 года)
		5-8	месяц (1-12)
		0-4	число (1-31)

День недели не записывается; DOS вычисляет его по остальной информации. Отметим также, что, как обычно, младший байт двухбайтовых значений расположен в памяти раньше, чем старший.

Средний уровень

Метод доступа к файлу с использованием управляющего блока файла позволяет получить дату последнего доступа к файлу, но не время. Когда FCB открывается функцией 0FH прерывания 21H, то заполняется двухбайтовое поле даты в приведенном выше формате. Это поле расположено в FCB со смещением 14H (см. [5.3.5]).

С другой стороны, доступ к файлу с помощью дескриптора файла позволяет как получить, так и установить дату и время последнего доступа к файлу. Функция 57H прерывания 21H выполняет все операции. Для установки времени и даты поместите номер файла в BX и 0 в AL. Для получения даты и времени надо поместить в AL 1. В обоих случаях дата содержится в DX, а время - в CX. Значение битов совпадает с тем, что дано в таблице. В техническом руководстве по MS-DOS утверждается, что младшие байты информации находятся в CH и DH, и наоборот. На самом деле это не так. При возникновении ошибки устанавливается флаг переноса, а в AX возвращается 1, если в AL указано неправильное число, и 6, если неверен дескриптор файла. В следующем примере определяется час, в который был отмечен последний доступ к файлу:

```
;---в сегменте данных
PATH DB   'B:NEWDATA.BAK',0
;---открываем файл
```

```
        LEA   DX,PATH        ;указываем на строку пути
        MOV   AH,3DH         ;функция открытия файла
        MOV   AL,0           ;открываем для чтения
        INT   21H            ;открываем файл
        JC    OPEN_ERROR     ;переход на обработку ошибки
;---получаем дату и время доступа к файлу
        MOV   BX,AX          ;помещаем номер файла в BX
        MOV   AL,0           ;код для чтения времени
        MOV   AH,57H         ;номер функции
        INT   21H            ;получаем время доступа
        JC    TIME_ERROR     ;переход на обработку ошибок
;---сдвигаем биты, относящиеся к часам, в начало CH
        MOV   CL,3           ;готовим сдвиг
        SHR   CH,CL          ;теперь CH содержит час доступа
```

5.2.6. Спрятанные и защищенные от записи файлы

DOS использует шесть различных атрибутов файлов, которые дают данному файлу определенный статус. Файл может иметь несколько атрибутов одновременно (но не все). Атрибуты устанавливаются 12-м байтом 32-байтового элемента каталога. Значение имеют младшие шесть битов, а остальные должны быть равны нулю. Биты таковы:

если бит	5 = 1,	файл был изменен со времени последней архивации
	4 = 1,	файл является подкаталогом
	3 = 1,	этот элемент является не файлом, а меткой тома
	2 = 1,	файл является "системным"
	1 = 1,	файл спрятан при поиске по каталогу
	0 = 1,	файл объявлен только для чтения

Бит *5* - это архивный бит, используемый программами BACKUP и RESTORE DOS. Этот бит сбрасывается в 0 после архивации и устанавливается, когда с файлом снова работали. При следующей архивации неизмененные файлы могут быть обнаружены и проигнорированы.

Высокий уровень

Бейсик не позволяет вам напрямую устанавливать атрибуты файла. Справьтесь в [5.2.1], как считать каталог в память, найти нужный файл, сделать изменения и снова записать его на диск.

Как только каталог помещается в память, байты атрибутов находятся по смещениям 11, 43, 75 и т.д. При необходимости вы можете прочитать текущие атрибуты и изменить только один бит, используя технику битовых операций, описанную в приложении Б. Но легче просто переписать все атрибуты заново. Будьте внимательны, ошибки могут быть фатальными. В данном примере считываются атрибуты файла с именем "NEWDATA.AAA":

```
100 'читаем сектора каталога, начиная с &H2000 и затем ...
110 DEF SEG = &H2000                    'указываем на область каталога
120 FILENAME$ = "NEWDATAAAA"            'ищем имя файла без точки
130 DIRPTR = 0                          'указатель в каталоге
140 FOR N = 1 TO 112                    'проверяем все элементы
150 X$ = ""                             'временная строка для имени файла
160 FOR M = 0 TO 10                     'для каждого символа имени
170 X$ = X$ + PEEK(DIRPTR + M)          'добавляем его к строке
180 NEXT
190 IF X$ = FILENAME$ THEN 220          'если имя найдено, выходим
200 NEXT
210 PRINT "File not found": END         'нет такого файла
220 X = PEEK(DIRPTR + 11)               'получаем атрибуты нужного файла
230 IF X AND 32 <> 0 THEN PRINT "File not baked up"
240 IF X AND 16 <> 0 THEN PRINT "File is a subdirectory"
250 IF X AND 8 <> 0 THEN PRINT "Volume label - not a file"
260 IF X AND 4 <> 0 THEN PRINT "File is a system file"
270 IF X AND 2 <> 0 THEN PRINT "File is a hidden file"
280 IF X AND 1 <> 0 THEN PRINT "File is read-only"
```

Средний уровень

Функция 43H прерывания 21H может как находить, так и изменять атрибуты файла, но только если файл был открыт с помощью метода дескриптора файлов, а не с помощью метода управляющего блока файла. Аналогичной функции для FCB нет. Байт атрибутов может быть установлен при создании файла (см. [5.3.2]) при использовании расширенного управляющего блока файла. Но если вы последовательно откроете FCB, измените установку атрибутов и затем закроете файл, то у него останутся первоначальные атрибуты. Хотя, конечно, вы можете изменить атрибуты каким-нибудь обходным путем, но намного проще применить функцию, использующую метод дескриптора файлов.

Чтобы присвоить файлу байт атрибутов, содержащийся в CX (на самом деле в CL, поскольку CH равен 0), поместите 1 в AL. Можно, наоборот, поместить в AL 0, чтобы в CX был возвращен текущий байт атрибутов файла. В обоих случаях DS:DX должны указывать на строку, представляющую путь к файлу. Конец строки отмечается байтом ASCII 0 (который не входит в число 63 символов). В предлагаемом примере статус "hidden" (спрятанный) присваивается файлу OVERDUE:

```
;---в сегменте данных
PATH DB  'A:ACCOUNTS',0
;---включаем признак спрятанного файла
     MOV AH,43H          ;номер функции
     MOV AL,0            ;читаем байт атрибутов
     LEA DX,PATH         ;DS:DX указывают на путь
     INT 21H             ;байт атрибутов в CX
     JC  ERROR_ROUTINE   ;обработка ошибок
     OR  CL,10B          ;включаем бит 1
     MOV AH,43H          ;номер функции
     MOV AL,1            ;заменяем байт атрибутов
     INT 21H             ;теперь файл стал спрятанным
```

Флаг переноса устанавливается при возникновении ошибки. В этом случае в AX возвращается 2, если файл не найден, 3, если не найден путь, и 5 - при других ошибках (нет доступа).

5.2.7. Чтение/изменение метки тома

Метка тома для дискеты - это элемент каталога, имеющий специальный атрибут. Метка занимает первые 11 байт элемента, относящиеся к имени и расширению файла. Байт атрибутов по смещению 11 содержит значение 8 (бит 3 = 1). Поля времени и даты заполняются обычным образом. Одним из свойств этого атрибута является то, что данный элемент не выводится по команде DIR.

Метка может занимать любую позицию в каталоге. Она ищется перебором всех байтов атрибутов, пока не будет найдено значение 8. Чтобы стереть метку, надо просто поместить E5 в первый байт соответствующего элемента; сам байт атрибутов можно не менять. Для изменения метки необходимо записать новые 11 символов (остаток заполняется пробелами). Если вам понадобится присвоить метку тома диску, который ее не имел, то

надо только найти пустое место в каталоге и записать туда метку и соответствующий атрибут.

Высокий уровень

Обсуждение в [5.4.2] объясняет, как читать и писать абсолютные сектора в Бейсике. Для стандартной двухсторонней дискеты надо использовать номер стороны 0, номер дорожки 0, номер сектора 6 и число секторов для чтения/записи - 7. После того как данные записаны в отведенный буфер, для изменения или добавления метки тома можно воспользоваться примерами, приведенными здесь. Затем сектора должны быть перезаписаны на диск. Будьте внимательны, ошибка может привести к потере всей информации на диске. В следующем примере показаны поиск метки тома и ее изменение:

```
100 'сектора загружены, начиная, скажем, с &H1000
110 DEF SEG = &H1000
120 DIRPTR = 11                          'указатель на байт атрибутов
130 FOR N = 1 TO 112                     'проверяем все элементы каталога
140 IF PEEK(DIRPTR) = 8 THEN 180         'выход, если метка тома
150 DIRPTR = DIRPTR + 32                 'указываем на следующий элемент
160 NEXT                                 'проверяем его атрибут
170 PRINT "No volume label found": END   'метки нет
180 INPUT "Enter new volume label", V$   'запрос метки
190 IF LEN(V$) > 11 THEN BEEP: PRINT "11 chars only": GOTO 180
200 V$ = V$ + STRING$(11-LEN(V$),32) 'дополняем пробелами
210 DIRPTR = DIRPTR - 11                 'возвращаемся на начало элемента
220 FOR N = 1 TO LEN(V$)                 'помещаем все символы метки
230 POKE N,MID$(V$,N,1)                  'в память
240 NEXT
250 'теперь осталось перезаписать сектора на диск
```

Низкий уровень

В приведенном ниже примере предполагается, что вы создали буфер данных размером 3584 байт для хранения всех семи секторов каталога дискеты емкостью 360K. Буфер называется DIR_AREA. В первом примере метка тома ищется и выводится или, если она не найдена, выводится сообщение об ее отсутствии. Для удобства область буфера отводится в сегменте данных; лучше отвести память для задачи, а затем освободить ее (см. [1.3.1]).

```
;---в сегменте данных
VOL_STR  DB    'The volume label is $'
NO_LAB   DB    'There is no volume label $'
DIR_AREA DB    3584 DUP(?)
;---читаем 7 секторов каталога
         MOV   AX,SEG DIR_AREA    ;сегмент буфера
         MOV   ES,AX
         MOV   BX,OFFSET DIR_AREA      ;смещение буфера
         MOV   DL,0               ;номер накопителя
         MOV   DH,0               ;номер головки
         MOV   CH,0               ;номер дорожки
         MOV   CL,6               ;стартовый сектор
         MOV   AL,7               ;число секторов/каталога
         MOV   AH,2               ;номер функции чтения
         INT   13H                ;читаем каталог в память
;---ищем метку тома, сравнивая байт атрибутов с 8
         MOV   CX,112             ;число элементов
         ADD   BX,11              ;смещение для атрибутов
TRY_AG:  MOV   AL,[BX]            ;берем 1-й элемент
         CMP   AL,8               ;это метка тома?
         JE    GOT_IT             ;если да, то выход
         ADD   BX,32              ;иначе, на следующий элемент
         LOOP  TRY_AG
;---выводим сообщение об отсутствии метки тома
         MOV   AH,9               ;функция вывода строки
         LEA   DX,NO_LAB          ;указываем на строку
         INT   21H                ;выводим ее
         JMP   SHORT CONT         ;на конец
;---выводим строку, выдающую метку тома
GOT_IT:  MOV   AH,9               ;функция вывода строки
         LEA   DX,VOL_STR         ;указываем на строку
         INT   21H                ;выводим ее
         SUB   BX,11              ;указатель на метку
         MOV   CX,11              ;пишем 11 символов
         MOV   AH,2               ;функция вывода символов
NEXT_CH: MOV   DL,[BX]            ;символ в DL
         INT   21H                ;выводим символ
         INC   BX                 ;переходим к следующему
         LOOP  NEXT_CH
CONT:
```

Чтобы стереть метку, поместите следующий код в GOT_IT:

```
GOT_IT: MOV AL,0E5H   ;код отметки пустого элемента
        SUB  BX,11    ;устанавливаем указатель на начало элемента
        MOV [BX],AL   ;меняем первый байт
```

Для того чтобы изменить метку тома, необходимо воспользоваться в GOT_IT следующим кодом (предполагается, что вы уже подготовили 11-байтовую строку NEW_LAB):

```
GOT_IT: LEA SI,NEW_LAB    ;SI должен указывать на строку
        SUB BX,11         ;BX указывает на начало метки
        MOV DI,BX         ;помещаем указатель в DI
        MOV CX,11         ;пересылка 11 символов
REP     MOVSB             ;пересылаем строку
```

Для создания метки применяют тот же самый код, но в этом случае надо установить байт атрибутов равным 8. (Вы можете просто добавить ASCII 8 к строке, содержащей новую метку, так как байт атрибутов следует непосредственно за самой меткой.)

И наконец, во всех случаях изменения каталога необходимо записать его обратно на диск. При этом ошибки допускать нельзя.

```
;---запись измененных секторов обратно на диск
MOV  AX,SEG DIR_AREA   ;регистры, как и при чтении
MOV  ES,AX
MOV  BX,OFFSET DIR_AREA
MOV  DL,0
MOV  DH,0
MOV  CH,0
MOV  CL,6
MOV  AL,7
MOV  AH,3              ;номер функции записи секторов
INT  13H
```

5.3. Подготовка к работе с файлами

В программах, написанных на языках высокого уровня, можно легко открыть файл - вся подготовительная работа для операций с файлами в этом случае будет выполнена

автоматически. Однако программисты на языке ассемблера должны создать специальные области данных, которые используются при операциях ввода/вывода. MS-DOS использует два метода доступа к файлам: метод управляющего блока файла (FCB) и метод дескриптора файла. Метод FCB существует с тех пор, когда MS-DOS еще не работала с древовидной структурой каталогов, поэтому с его помощью можно получить доступ только к файлам, находящимся в текущем каталоге. Метод дескриптора файла позволяет получить доступ к любому файлу, независимо от того, какой каталог является текущим.

Поскольку теперь древовидная структура каталогов широко используется, то метод FCB становится анахронизмом, но MS-DOS продолжает поддерживать этот метод, чтобы сохранить совместимость со старым программным обеспечением, и по этой причине мы рассмотрим и его. Однако в своих программах всегда старайтесь применять метод дескриптора файла. Метод дескриптора файла имеет дополнительное преимущество в том, что он требует меньше подготовительной работы, хотя в некоторых приложениях сами операции ввода/вывода при этом могут оказаться более сложными, чем в методе FCB. Например, операции чтения файла с прямым доступом с использованием метода дескриптора файла требуют, чтобы программа вычисляла смещение каждой записи в файле, в то время как соответствующая функция FCB получает номер записи и делает необходимые вычисления сама.

Прежде чем читать или писать данные в файл, его нужно открыть. Открыть файл это значит создать и инициализировать специальную область данных, используемую MS-DOS, которая содержит важную информацию о файле, такую, как имя файла, имя накопителя, размер записи файла и т.д. Языки высокого уровня, такие, как Бейсик, создают эти области автоматически. Одной из таких областей является управляющий блок файла и, когда используется метод FCB, программа создает этот блок, а MS-DOS читает и манипулирует его содержимым. Первоначально FCB содержит только имя файла и имя накопителя; после того как файл открывается, в него добавляется информация о размере записи и текущей позиции, с которой к нему будет осуществляться доступ.

При доступе к файлам с помощью дескриптора файла MS-DOS автоматически создает область данных для файла в произвольном месте. Затем MS-DOS создает уникальный 16-битовый код номера файла; впоследствии этот "номер" используется функциями DOS для идентификации того, с каким из открытых файлов производится операция. Все, что нужно для нахождения файла, - это стандартная строка пути, в которой может быть необязательное имя накопителя, а имена подкаталогов разделены обратной косой

чертой. Эти строки отличаются от стандартного запроса MS-DOS только тем, что они должны завершаться байтом ASCII 0, чтобы программа могла найти конец строки (такие строки называются строками ASCIIZ).

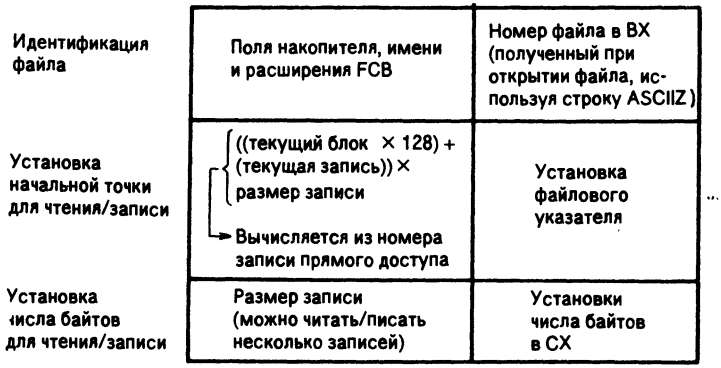


Рис. 5.2. Два метода доступа к файлам

Операции по пересылке данных из файла или в файл требуют, чтобы была указана область памяти, в которую или из которой будут направляться данные. Такой буфер определяется отведением ему места в памяти и установкой указателя на его первый байт (т.е. на младший адрес буфера в памяти). Если передано слишком много данных, то буфер переполняется и может разрушить данные, расположенные в следующих адресах памяти. Буфер может использоваться как промежуточный, работающий только с небольшой порцией данных для операций чтения или записи. Буфер также может помещаться в область памяти, в которой программа действительно хранит и обрабатывает данные.

Функции доступа через управляющий блок файла определяют промежуточный буфер с помощью указателя, который все время хранится операционной системой. Этот буфер называется *областью обмена с диском* (disk transfer area) или *DTA*. К сожалению, в технической документации по IBM PC часто термином DTA называют указатель на буфер, хотя на самом деле правильно называть его указателем на DTA. После того как указатель на DTA установлен с помощью специальной функции, все файловые операции используют его до тех пор, пока он не будет изменен. Функции же, использующие метод дескриптора файла, должны указывать стартовый адрес буфера обмена каждый раз при вызове

функции, указатель на DTA они игнорируют, в отличие от функций управляющего блока файла. Рис. 5.2 показывает два метода доступа к файлам.

5.3.1. Установка/проверка накопителя по умолчанию

Программы могут экономить часть работы, назначая накопитель, на котором содержатся файлы данных, по умолчанию. Если в начале выполнения программа запросит у пользователя, с каким накопителем он будет работать, то впоследствии у нее не будет сомнений, к какому накопителю следует обращаться.

Высокий уровень

В приведенной программе на Бейсике текущий накопитель по умолчанию переключается с помощью процедуры на машинном языке. Процедура имеет длину 7 байт. Она помещается в строку X$, а переменная Z служит указателем на первый байт процедуры. В приложении Г объясняется, как вставлять ассемблерные процедуры в программы на Бейсике. Номер накопителя устанавливается в строке 110, причем 0 = A, 1 = B и т.д. Если назначенный по умолчанию накопитель - несуществующий, то ошибки не будет, поэтому будьте внимательны. Не пытайтесь объединить строки 120 и 130 этой процедуры, поскольку в этом случае интерпретатор Бейсика будет обрабатывать их неправильно.

```
100 DEF SEG                          'сегмент на начало области Бейсика
110 NUM = 0                          'выбираем накопитель A
120 X$ = CHR$(180) + CHR$(14) + CHR$(178) + CHR$(NUM) +
CHR$(205) + CHR$(33) + CHR$(223)
130 Y = VARPTR(X$)                   'получаем дескриптор строки (адрес в Y + 1)
140 Z = PEEK(Y + 1) + PEEK(Y + 2)*256        'вычисляем адрес строки
150 CALL Z                           'выполняем машинную процедуру
```

Средний уровень

Функция EH прерывания 21H устанавливает накопитель по умолчанию. Надо просто поместить номер накопителя (0 = A, 1 = B и т.д.) в DL и выполнить прерывание. Эта функция возвращает в AL число накопителей на машине. Отметим, что когда у машины имеется только один накопитель, то возвращается число 2. Лучший способ определения числа накопителей у машины описан в [1.1.5].

```
MOV  AH,0EH        ;номер функции
MOV  DL,1          ;код для накопителя B
INT  21H           ;устанавливаем накопитель по умолчанию
```

Функция 19H прерывания 21H сообщает, какой из накопителей является накопителем по умолчанию. Для этой функции нет входных регистров. При возврате в AL содержится кодовый номер, где 0 = A, 1 = B и т.д.

5.3.2. Создание/удаление файла

Можно создать файл, не помещая в него никакой информации. В этом случае создается элемент каталога, а длина файла устанавливается равной 0. При удалении файла соответствующий элемент каталога на самом деле не удаляется, он просто становится недействующим за счет замены первого байта элемента (первого символа имени файла) на E5H. Впоследствии этот элемент может быть перезаписан при создании нового файла. Во время удаления файла вносятся также изменения в таблицу размещения файлов, с тем чтобы сектора, используемые этим файлом, были доступны для других файлов. Само содержимое секторов при этом не стирается. Поэтому можно восстановить удаленный файл, однако предупреждаем, что операции с таблицей размещения файлов надо производить очень осторожно.

Высокий уровень

Бейсик не имеет специальной команды для создания файла. Вместо этого при открытии файла указанное имя ищется в каталоге, и если оно не найдено, создается новый файл. Если открыть новый файл, а затем закрыть его, не производя в него записи, то он останется в каталоге с длиной 1 байт, и ему будет отведен кластер дискового пространства (единственный байт - символ Ctrl-Z - ASCII 26, который используется в качестве признака конца стандартного текстового файла). Детали оператора OPEN см. в [5.3.3].

Оператор CLOSE файл не уничтожает, для этой цели служит оператор KILL. Для того чтобы уничтожить файл, не надо его открывать. Просто поместите имя файла в кавычках, например KILL "A:ACCOUNT.DAT". Или, если файл находится в другом подкаталоге, надо использовать стандартный путь к файлу, например KILL "A:\FINANCES\ACCOUNT.DAT". В обоих случаях имя накопителя необходимо указывать, только если файл находится не на накопителе, принятом по умолчанию. Отметим, что

вы не можете воспользоваться этим методом, чтобы удалить подкаталог (который является одним из видов файла), - в этом случае применяйте RMDIR.

Средний уровень

Файл может быть создан или уничтожен с помощью либо метода управляющего блока файла, либо метода дескриптора файла. Создание файла одним из перечисленных методов ни в коей мере не ограничивает будущий доступ к нему только этим методом. Но поскольку одновременно с созданием он открывается, то при создании необходимо использовать тот метод, с помощью которого вы будете работать с этим файлом на этот раз. Когда файл создается, а затем закрывается и при этом в него ничего не записывается, ему соответствует элемент каталога с нулем в поле длины файла, однако дисковое пространство этому файлу не отводится. Важно понимать, что когда последовательный файл открывается для записи (а не для добавления) данных, то используется именно эта функция создания файла, поэтому файл обрезается до нулевой длины и затем полностью перезаписывается.

Метод FCB. Функция 16H прерывания 21H создает и открывает файл. Создайте FCB с именем файла и накопителя и пусть DS:DX указывают на него. Затем вызовите функцию. Если во время просмотра каталога найден совпадающий элемент, то он будет повторно использован для записи информации о создаваемом файле, уничтожая тем самым сведения о старом файле. Чтобы избежать непреднамеренного разрушения файлов, сначала проверьте наличие файла с таким именем с помощью функции 11H прерывания 21H (см. [5.2.1]). Если в каталоге файла с таким именем нет, то создается новый элемент и в AL возвращается 0, если каталог полон, то - FF. Чтобы присвоить файлу специальные атрибуты (например, статус только для чтения [5.2.6]), используйте расширенный управляющий блок файла [5.3.5]. При открытии новый файл инициализируется с нулевой длиной и ему отводится кластер дискового пространства. Например:

```
;---в сегменте данных
FCB  DB   1,'MYFILE DAT',25 DUP(0)
;---проверка наличия такого файла
     MOV  AH,11H        ;функция поиска файла
     LEA  DX,FCB        ;DS:DX указывают на FCB
     INT  21H           ;ищем файл
     CMP  AL,0          ;AL = 0, если файл существует
```

```
    JE    WARN_USER  ;если да, то сообщаем об этом
;---создание файла
    MOV AH,16H       ;номер функции создания файла
    INT  21H         ;создаем файл
```

Для создания файла со специальными атрибутами, например статусом только для чтения, надо использовать расширенный управляющий блок файла. (Байт атрибутов файла обсуждался в [5.2.6].) К обычному FCB надо добавить 7-байтовый заголовок, начиная с байта FFH, далее следуют 5 байт ASCII 0, а затем нужный байт атрибутов. Для создания спрятанного файла необходимо, чтобы был установлен бит 1 байта атрибутов. Чтобы спрятать файл, открытый в приведенном примере, напишите:

```
FCB   DB  0FFH,5 DUP(0),2,1,'MYFILE DAT',25 DUP(0)
```

Функция 13H прерывания 21H уничтожает файл. В этом случае надо, чтобы DS:DX указывали на неоткрытый FCB и только затем выполните функцию. Если файл с указанным именем не найден, то в AL возвращается FF, иначе - 0. В имени файла могут использоваться "джокеры" (знаки вопроса, но не "звездочки"), тогда за одно обращение к функции может быть удалено несколько файлов. Например:

```
;---в сегменте данных
FCB DB   1,'MYFILE DAT',25 DUP(0)
;---удаляем файл
    MOV AH,13H       ;номер функции удаления файла
    LEA  DX,FCB      ;DS:DX указывают на FCB
    INT  21H         ;удаляем файл
    CMP AL,0FFH      ;проверка на ошибку
    JE   DEL_ERROR   ;уход на обработку ошибки
```

Метод дескриптора файла. Функция 3CH прерывания 21H создает и открывает новый файл методом дескриптора файла. DS:DX должны указывать на строку, представляющую путь к файлу и имя файла в стандартном формате MS-DOS, включая имя накопителя, если файл находится не на накопителе, принятом по умолчанию. Строка должна завершаться байтом ASCII 0. Байт атрибутов файла (см. [5.2.6]) поместите в CX (0 - для обычного файла). Затем выполните функцию. При успешном выполнении флаг переноса будет равен нулю, а в AX будет возвращен номер нового файла. При возникновении ошибки флаг переноса устанав-

ливается в 1, а в AX содержится код ошибки, который может быть равен 3, если не найден путь, 4, если уже открыты все буфера для файлов, и 5, если каталог полон или файл уже существует со статусом только для чтения. Отметим, что если в каталоге уже существует файл с таким именем, то он "обрезается" до нулевой длины, и тем самым разрушается. Во избежание ошибок надо предварительно использовать функцию 4EH прерывания 21H для проверки.

```
;---в сегменте данных
PATH DB   'B:LEVEL1\LEVEL2\FILENAME.EXT',0
;---проверка наличия файла в каталоге
      MOV AH,4EH          ;функция поиска в каталоге
      LEA  DX,PATH        ;DS:DX указывают на путь
      INT  21H            ;проверка наличия файла
      JNC  WARN_USER      ;если есть, то сообщаем
;---создание файла
      MOV AH,3CH          ;функция создания файла
      MOV CX,0            ;нормальные атрибуты
      INT  21H            ;создаем файл
      JC   OPEN_ERROR     ;уход на обработку ошибки
      MOV HANDLE,AX       ;запоминаем номер файла
```

В MS-DOS версии 3.0 добавлена новая функция для создания файла методом дескриптора файла. Это функция 5BH прерывания 21H. Она работает точно так же, как и описанная выше функция 3CH, за исключением того, что она возвращает расширенные коды ошибок, что позволяет лучше обрабатывать ошибочные ситуации. Они объяснены в [7.2.5].

Для уничтожения файла методом дескриптора файла используйте функцию 41H прерывания 21H. И опять DS:DX должны указывать на строку, дающую путь к файлу, и его имя. "Джокеры" в имени файла не допускаются. Затем вызовите функцию. Если при возврате флаг переноса установлен, то функция не выполнена, в этом случае AL будет содержать 2, если файл не найден, и 5, если произошла ошибка на диске. Отметим, что с помощью этой функции вы не можете удалить файл со статусом только для чтения; измените атрибуты файла (см. [5.2.6]) перед его удалением. Например:

```
;---в сегменте данных
PATH DB   'B:LEVEL1\LEVEL2\FILENAME.EXT',0
;---уничтожаем файл
      MOV AH,41H          ;номер функции уничтожения
```

```
LEA  DX,PATH          ;DS:DX указывают на путь
INT  21H              ;уничтожаем файл
JC   DELETE_ERROR     ;уход на обработку ошибки
```

MS-DOS версии 3.0 имеет специальную функцию (5AH прерывания 21H) для создания временного "безымянного" файла. Операционная система сама генерирует имя для файла и проверяет, что такого файла еще нет в каталоге. При этом исключается всякая возможность того, что при создании временного файла будет разрушен существующий файл с совпадающим именем. При входе DS:DX должны указывать на строку пути к каталогу, в котором должен быть создан временный файл. Строка завершается обратной косой чертой. Поместите атрибуты файла в CX (обычно 0). При возврате AX будет содержать номер файла, если только флаг переноса не установлен, в этом случае AX содержит информацию об ошибке. Произвольное имя файла добавляется к концу строки пути. Эта функция может возвращать расширенные коды ошибок, которые существуют только в MS-DOS 3.0; они объясняются в [7.2.5]. Файл, созданный этой функцией, не уничтожается автоматически - программа должна использовать функцию 41H (см. выше). В предлагаемом примере программа создает временный файл, а затем уничтожает его:

```
;---в сегменте данных
PATH DB  'B:LEVEL1\LEVEL2\',12 DUP(0)
;---создаем временный файл
     MOV AH,5AH       ;номер функции
     LEA DX,PATH      ;DS:DX указывают на путь
     INT 21H          ;создаем временный файл
     JC  CRE_ERROR    ;уход на обработку ошибки

     MOV AH,41H       ;номер функции
     LEA DX,PATH      ;DS:DX указывают на путь
     INT 21H          ;уничтожаем временный файл
     JC  DEL_ERROR    ;уход на обработку ошибки
```

5.3.3. Открытие/закрытие файла

"Открыть" файл - это значит создать небольшие блоки памяти, где будет содержаться информация о файле. Они будут служить промежуточной станцией (буфером), через которую данные будут передаваться между файлом и памятью. Языки высокого уровня

автоматически создают для вас эти блоки, а язык ассемблера - нет. При открытии файла каталог проверяется на его наличие. Если файл найден, то MS-DOS берет информацию из каталога о размере и дате создания файла. Затем, при закрытии файла, система обновляет информацию в каталоге. Закрытие файла также очищает все системные буфера переноса, посылая на диск оставшуюся информацию. Если вы не закроете файл перед завершением программы, то это может привести к потере данных.

Когда программа работает со многими файлами, надо постоянно помнить о том, сколько файлов открыто одновременно. MS-DOS 2.1 позволяет иметь до 99 одновременно открытых файлов, причем по умолчанию - только 8 (измените это число с помощью команды MS-DOS FILES). Бейсик позволяет одновременно открывать не более 15 файлов. Каждый файл занимает место для блока параметров и буфера. Память для каждого файла отводится отдельно перед тем, как он открывается, и потому недоступна для программ, даже если указанное число файлов не используется в настоящий момент. По этой причине вы можете экономить память с помощью описанного метода, устанавливая максимально допустимое число открытых файлов именно таким, которое вам требуется.

Высокий уровень

При открытии файла в Бейсике идет поиск в каталоге, и если файл не найден, то создается новый файл с данным именем. Имеются два способа записи оператора открытия файла и в большинстве случаев они эквивалентны. Единственное отличие состоит в том, что один из них более закодирован, в то время как другой ближе к естественному языку. В обоих операторах вы должны указать по меньшей мере три вида информации. Во-первых, требуется имя файла, и поскольку это строка, оно должно быть заключено в кавычки. Во-вторых, число, начиная с 1, присваивается файлу как идентификационный номер, по которому другие операторы читают или пишут в файл. И в-третьих, вы должны указать, для какой цели открывается данный файл, т.е. открыт ли он для прямого или для последовательного доступа. Чтобы открыть файл MYFILE.TXT для записи в последовательный файл (файл будет иметь номер 2), запишите или

```
OPEN "O",#2,"MYFILE.TXT"
```

или

```
OPEN "MYFILE.TXT" FOR OUTPUT AS #2
```

Отметим, что в обоих случаях номер 2 относится к буферу файла #2. Число может быть любым, не превосходящим количества разрешенных буферов для файлов. Если поддерживается одновременная работа с шестью файлами, то число должно быть в интервале от 0 до 6. Однако буфер файла с номером 1 не обязательно использовать раньше, чем файла с номером 2. По умолчанию Бейсик устанавливает число буферов, равное 8, но вы можете изменить его на другое (от 4 до 15). Из этих файлов четыре используются Бейсиком для своих нужд, поэтому по умолчанию только 4 файла доступны вам для ввода/вывода. Для того чтобы установить число доступных буферов, используйте параметр F: при запуске Бейсика. Например, если вы при старте Бейсика напишете BASICA/F:10, то будет создано 10 буферов, шесть из которых доступны для файловых операций.

Второй параметр, S:, устанавливает размер буферов файла. Все буфера имеют один и тот же размер. По умолчанию берется размер, равный 128 байтам, однако допустимы размеры до 32767 байт. Для файлов последовательного доступа этот размер может быть установлен равным 0, что позволяет немного сэкономить память. Для файлов прямого доступа он должен быть не меньше максимального размера записи. Отметим, что если размер записи равен 512 байтам и размер буфера также равен 512 байтам, то это приводит к ускорению дисковых операций. Команда BASICA/S:512/F:10 открывает 10 буферов размером 512 байт. Каждый файл требует 188 байт плюс размер буфера, поэтому для такой конфигурации потребуется 7К памяти. Число буферов не может быть больше, чем разрешено иметь открытых файлов в DOS.

Кодированная форма. Первая из форм оператора OPEN использует одну букву для обозначения желаемого типа операций над файлом. Имеется три возможности:

"O" - открыть файл с последовательным доступом для вывода;

"I" - открыть файл с последовательным доступом для ввода;

"R" - открыть файл с прямым доступом для ввода/вывода.

Последовательные файлы не могут записываться, когда они открыты для чтения, и наоборот. В типичных случаях последовательные файлы открываются для чтения, затем считываются целиком в память и закрываются. После того как необходимые изменения внесены, файл снова открывается, но теперь для вывода, и записывается обратно на диск, перекрывая то, что было записано в его секторах и, возможно, захватывая новые сектора.

Следует отметить несколько моментов, относящихся к этой форме оператора OPEN. Имя файла должно содержать имя накопителя, если файл не найден на накопителе, принятом по умолчанию (т.е. накопителе, с которого запущен Бейсик). Имя файла может также содержать путь к файлу, находящемуся в подкаталоге, например OPEN "I",#1,"A:\LEVEL1\LEVEL2\MYFILE.TXT". Кроме того, вы можете поместить указание размера записи в конце оператора OPEN "R",#3,"MYFILE.TXT",52. В этом случае каждая запись будет занимать 52 байта дискового пространства. Если в операторе FIELDS не будут задействованы все 52 байта, то оставшиеся пропадут. Этот параметр существен при операциях с файлами прямого доступа. Большинство операций с файлами последовательного доступа не требует указания длины записи, однако вы можете ускорить файловые операции, установив размер записи равным 512 байтам. Длина записи может быть в диапазоне от 1 до 32767 байт и по умолчанию равна 128 байтам.

Форма естественного языка. Вторая форма оператора OPEN делает совершенно то же самое, что и первая, но использует полные слова. Вместо того чтобы писать "O" или "I", вы должны писать INPUT или OUTPUT (без кавычек), например, OPEN "FILENAME" FOR INPUT AS #1. Для файлов с прямым доступом не указывается этот параметр: OPEN "MYFILE.TXT" AS #2. Кроме того, вы можете указать режим APPEND, чтобы записать данные, начиная с конца последовательного файла, не уничтожая уже существующих: OPEN "B:MYFILE.TXT" FOR APPEND AS #3. Как и для первой формы, в операторе может быть указана необязательная длина записи. Надо просто добавить в конце оператора LEN = число. Например, OPEN "C:MYFILE.TXT" AS #1 LEN = 52 открывает файл прямого доступа с записями длиной 52 байта.

Часто программа должна запрашивать нужный файл у пользователя. Для его открытия в операторе OPEN просто подставьте вместо имени файла идентификатор строки, содержащей это имя. При этом необходима проверка на правильность введенного имени.

```
100 INPUT "Enter file name: ",F$              'получаем имя файла
110 IF INSTR(F$,".") <> 0 THEN 130            'есть ли расширение?
120 IF LEN(F$) > 8 THEN 500 ELSE 150          'длиннее 8 символов?
130 IF LEN(F$) > 12 THEN 500                  'длиннее 12 символов?
140 IF LEN(F$) - INSTR(F$,".") > 3 THEN 500 'тип длиннее 3
150 OPEN F$ FOR INPUT AS #1                   'открываем файл
```

```
500 INPUT "Improper filename - enter another: ",F$
510 GOTO 110                    'если имя неверное, новый запрос
```

Закрытие файла. Закрытие файла тривиально. Чтобы закрыть все открытые файлы, напишите CLOSE. Чтобы закрыть определенный файл или несколько файлов, напишите CLOSE #1 или CLOSE #1, #3. Важно закрыть все файлы перед завершением программы. Если этого не сделать, то в файле могут остаться данные, которые не будут записаны на диск. Отметим, что команды END, NEW, RESET, SYSTEM и RUN закрывают все буфера файлов, но не очищают их. Уже закрытый файл всегда может быть снова открыт с использованием любого доступного буфера файла.

Средний уровень

MS-DOS обеспечивает различные функции для открытия и закрытия файла, в зависимости от того, использовала ли программа для доступа к файлу метод управляющего блока файла или метод дескриптора файла. В обоих случаях могут быть открыты только файлы, которые существовали до этого. Для создания новых файлов применяют специальную функцию (см. [5.3.2]).

Метод FCB. Функция 0FH прерывания 21H открывает уже существующий файл. Вы должны сначала создать управляющий блок файла, как показано в [5.3.5]. Перед открытием FCB должен содержать только имя файла и имя накопителя (0 = по умолчанию, 1 = A и т.д.). DS:DX должны указывать на FCB. Затем надо выполнить функцию. При возврате AL будет содержать 0, если файл успешно открыт, и FF, если файл не найден. В случае, когда для указания накопителя используется 0, он будет заменен на код, соответствующий накопителю по умолчанию.

Только после того как файл открыт, вы должны установить размер записи (по умолчанию - 128 байт), поля записи прямого доступа и текущей записи (они обсуждаются в разделе, относящемся к операциям с последовательным и прямым доступом). При открытии поле текущего блока заполняется нулем, а поля даты и времени - информацией из каталога.

Чтобы закрыть файл с помощью метода FCB, надо установить DS:DX на открытый FCB и вызвать функцию 10H прерывания 21H. При успешном вызове информация о размере файла, дате и времени будет записана в каталог, а в AL будет возвращен 0. Однако, если имя файла не будет обнаружено в каталоге или оно будет найдено в другой позиции, изменения на диске будут индицированы возвратом FF в AL.

```
;---в сегменте данных
FCB DB    1,'FILENAMEEXT',25 DUP(0)
;---открытие файла
    MOV  AH,0FH          ;номер функции
    LEA  DX,FCB          ;DS:DX указывают на FCB
    INT  21H             ;открываем файл
    CMP  AL,0            ;проверка на ошибку
    JNE  OPEN_ERROR      ;уход на обработку ошибки

.
.
;---закрытие файла
    MOV  AH,10H          ;номер функции
    LEA  DX,FCB          ;DS:DX указывают на FCB
    INT  21H             ;закрываем файл
    CMP  AL,0            ;проверка на ошибку
    JNE  CLOSE_ERROR     ;уход на обработку ошибки
```

Метод дескриптора файла. Для открытия файлов используйте функцию 3DH прерывания 21H. DS:DX должны указывать на строку, представляющую путь к файлу, и его имя, включая имя накопителя, если это необходимо. Вся строка должна быть не длиннее 63 байт и завершаться символом ASCII 0. В AL надо поместить код доступа, причем 0 открывает файл для чтения, 1 - для записи, а 2 - для чтения/записи. При возврате AX будет содержать 16-битовый номер файла, по которому файл впоследствии идентифицируется. Файловый указатель устанавливается на начало файла. Размер записи устанавливается равным 1 байту - это связано с тем, что операции прямого доступа при использовании метода дескриптора файла не имеют специальных буферов (на самом деле файлы с прямым доступом рассматриваются как последовательные и с ними работают одни и те же функции). Эта функция позволяет открывать как обычные, так и спрятанные файлы. При возврате флаг переноса равен 0, если файл открыт успешно. В противном случае флаг переноса устанавливается, а AX содержит 2, если файл не найден, 4, если программа хочет открыть слишком много файлов, 6 - при ошибке на диске, и 12, если неправильно указан код доступа в AL. Например:

```
;---в сегменте данных
PATH DB   'A:LEVEL1\FILENAME.EXT',0
;---открываем файл для чтения/записи
      MOV AH,3DH          ;номер функции
```

```
MOV AL,2            ;открываем для чтения/записи
LEA DX,PATH         ;DS:DX указывают на путь
INT 21H             ;открываем файл
JC  OPEN_ERROR      ;уход на обработку ошибок
MOV HANDLE,AX       ;сохраняем номер файла
```

Функция 3EH прерывания 21H закрывает файл, открытый методом дескриптора файла. Надо просто поместить номер файла в BX и выполнить функцию. При возврате флаг переноса равен 0, если все в порядке, иначе он равен 1, а AX - 6, если указан неверный номер файла.

```
;---закрытие файла
MOV AH,3EH          ;номер функции
MOV BX,HANDLE       ;номер файла
INT 21H             ;закрываем файл
JC  CLOSE_ERROR     ;уход на обработку ошибки
```

Функция 45H прерывания 21H создает второй дескриптор файла из имеющегося открытого. В BX должен быть указан существующий номер, а в AX будет возвращен новый. Функция 46H прерывания 21H связывает второй дескриптор (помещаемый в CX) с открытым файлом (номер которого в BX) таким образом, что первый будет относиться к тому же файлу и устройству, что и последний.

5.3.4. Переименование файла; изменение позиции файла в каталоге

Переименование файла может заключаться лишь в изменении первых 11 символов элемента каталога. Однако в древовидном каталоге весь элемент каталога может быть перенесен в другой подкаталог, переопределяя тем самым путь к файлу. Достаточно одной команды, чтобы переименовать файл или перенести его в другой каталог.

Высокий уровень

В Бейсике файл переименовывается командой NAME. С помощью этой команды он может быть также перенесен в другой каталог. Напишите сначала существующее имя, а затем новое имя файла, оба заключенные в кавычки, например NAME

"OLDFILE.EXT" AS "NEWFILE.EXT". В этом случае будет переименован файл в корневом каталоге. Для изменения имен файлов, расположенных в подкаталогах, могут быть использованы пути к файлу. Например, NAME "B:LEVEL1\OLDFILE.EXT" AS "B:LEVEL1\NEWFILE.EXT" изменяет имя файла в подкаталоге LEVEL1.

Отметим, что для нового имени файла должен быть указан полный путь. Если вы запишете NAME "B:LEVEL1\OLDFILE.EXT" AS "NEWFILE.EXT", то файл будет не только переименован, но и перенесен в корневой каталог. Для переноса файла из одного подкаталога в другой без изменения его имени введите команду NAME "A:SUBDIR1\OLDFILE.EXT" AS "A:SUBDIR2\OLDFILE.EXT". Таким методом нельзя перенести файл с диска на диск. Поскольку файлы, расположенные в разных каталогах, могут иметь одно и то же имя, то возможна ошибка при попытке переноса файлов с одинаковыми именами. В этом случае будет возвращен код ошибки 58 (см. [5.4.8]).

Средний уровень

MS-DOS может переименовывать файлы, используя как метод управляющего блока файла, так и метод дескриптора файла. Первый из них может применяться только к файлам, расположенным в текущем каталоге.

Метод FCB. Примените функцию 17H прерывания 21H. DS:DX должны указывать на открытый управляющий блок файла. Поместите новое имя файла в FCB, начиная со смещения 11H (это "резервная" область блока). Новое имя может использовать символ "?", в этом случае символы, находящиеся в этих позициях, не будут изменяться. При возврате, если новое имя уже существовало в каталоге, AL будет равно FF, иначе - 0. В следующем примере имя файла ACCOUNTS.DAT меняется на DEBTS.DAT.

```
;---в сегменте данных
FCB     DB   'FILENAMEEXT',25 DUP(0)
NEWNAM DB   'NEWNAME EXT',      ;11 символов нового имени
;---помещаем новое имя файла в переменную NEWNAM
        MOV SI,OFFSET NEWNAM    ;DS:SI указывают на новое имя
        MOV AX,SEG FCB          ;ES:DI указывают на FCB
        MOV ES,AX
        MOV DI,OFFSET FCB
        ADD DI,11H              ;начинаем со смещения 11H
        MOV CX,11               ;имя файла содержит 11 символов
REP     MOVSB                   ;переносим 11 байт
        LEA  DX,FCB             ;DS:DX указывают на FCB
```

```
MOV AH,17H              ;функция изменения имени
INT 21H                 ;изменяем имя
CMP AL,0FFH             ;проверка на ошибку
JE  RENAME_ERROR        ;уход на обработку ошибки
```

Метод дескриптора файла. Функция 56H прерывания 21H переименовывает и перемещает файлы. DS:DX должны указывать на строку, показывающую путь и имя переименуемого файла (до 63 символов) и завершающуюся символом ASCII 0. ES:DI должны указывать на вторую строку, которая дает новые имя и путь файла. Имена накопителей, если они присутствуют, должны совпадать. Если пути различны, то файл переносится в другой подкаталог. Чтобы перенести файл без переименования, надо во второй строке указать то же самое имя, но другой путь. При возврате, если произошла ошибка, устанавливается флаг переноса, а AX будет содержать 3, если один из путей не найден, 5 - при ошибке на диске и 17 - при попытке переноса между разными накопителями. В предлагаемом примере файл ACCOUNTS.DAT переносится из подкаталога GAINS в подкаталог LOSSES:

```
;---в сегменте данных
OLDPATH DB   'A:GAINS\ACCOUNTS.DAT',0
NEWPATH DB   'A:LOSSES\ACCOUNTS.DAT',0
;---изменение пути файла
        LEA  DX,OLDPATH         ;DS:DX указывают на старый путь
        MOV AX,SEG NEWPATH      ;ES:DI указывают на новый путь
        MOV ES,AX
        MOV DI,OFFSET NEWPATH
        MOV AH,56H              ;номер функции
        INT  21H                ;переносим файл
        JC   ERROR_ROUTINE      ;уход на обработку ошибки
```

5.3.5. Подготовка к файловым операциям

Языки высокого уровня, такие, как Бейсик, выполняют подготовительную работу для файловых операций автоматически. Однако программы на языке ассемблера имеют достаточно работы, перед тем как создать или открыть файл. Требования отличаются в зависимости от того, используется ли для доступа к файлу метод управляющего блока файла или метод дескриптора файла. Для обоих методов вам необходимо подготовить строку или блок параметров, указывающих на файл и буфер для переноса данных.

MS-DOS предоставляет различные наборы функций чтения/записи для обоих методов.

Средний уровень

Метод управляющего блока файла. Этот метод доступа к файлам требует, чтобы вы создали блок параметров, который первоначально должен содержать такую информацию, которая позволяет найти файл в каталоге. Хотя FCB имеет много полей, но только некоторые из них должны быть заполнены; остальные же MS-DOS заполняет информацией после того, как файл открывается. Отметим, что к началу FCB может добавляться специальное поле для создания расширенного FCB, который объясняется ниже. Вот структура FCB:

Накопитель (DB)	Число, определяющее, на каком накопителе будет искаться файл, 1 = A, 2 = B и т.д. Если указан 0, то берется накопитель по умолчанию, а затем система заменяет 0 на код этого накопителя.
Имя и расширение (11 байт)	Восьмибайтовое имя файла, выравненное по левому краю, должно быть дополнено пробелами (ASCII 32), если оно меньше 8 байт. То же относится и к трехбайтовому расширению. Между ними не должна стоять точка.
Текущий блок (DW)	DOS организует файлы блоками по 128 записей, пронумерованных от 0 до 127. Например, система рассматривает запись #129 файла прямого доступа как запись #0 блока #1 (отсчет как для записей, так и для блоков ведется с 0). В файлах нет специальных ограничителей ни для блоков, ни для записей. Вместо этого смещение для блоков и записей вычисляется исходя из длины записи, которая устанавливается следующим полем FCB.
Размер записи (DW)	Все функции MS-DOS, связанные с чтением или записью в файл, работают в терминах записи. Для файлов прямого доступа важно, чтобы размер записи был установлен равным размеру записей, помещенных в файл. Для последовательных файлов размер записи не столь важен, однако маленький размер записи будет замедлять дисковые операции. Поскольку размер сектора равен 512 байтам, то оптимальным размером записи будет эта же величина. Система

автоматически помещает значение по умолчанию 80H (128) в поле длины записи при открытии файла. Поэтому не забудьте установить это поле после открытия файла.

Размер файла (DD) — Размер указывается с точностью до байта. Это поле заполняется системой при открытии файла.

Дата файла (DW) — Дата записывается системой при открытии FCB. Ее формат приведен в [5.2.5].

Текущая запись (DB) — Текущая запись используется совместно с полем текущего блока. Записи нумеруются от 0 до 127. Запись прямого доступа #200, расположенная в блоке 1, имеет номер текущей записи, равный 71 ((200 - 128) - 1).

Номер записи прямого доступа (DD) — Вместо того чтобы требовать от программы вычисления текущего значения блока и записи для файла прямого доступа, MS-DOS делает эту работу сама. При операциях с файлами прямого доступа просто поместите номер записи в это четырехбайтовое поле. При выполнении операции с файлом прямого доступа MS-DOS поместит нужные значения в поля текущего блока и текущей записи. Помните, что старший байт расположен в старшей ячейке.

Связь между полями текущей записи, текущего блока и номера записи прямого доступа показана на рис. 5.3.

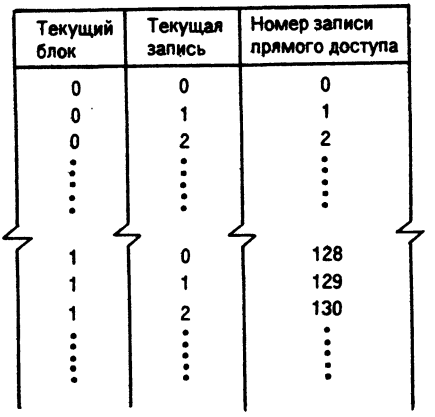


Рис. 5.3. Организация записей прямого доступа в методе FCB

Простейший путь - создать FCB как переменную в сегменте данных программы. Если имя открываемого файла не меняется, то оно может быть непосредственно записано в это поле. Остаток блока инициализируйте байтами ASCII 0. Только после того как FCB будет открыт (с помощью функции 0FH прерывания 21H, как показано в [5.3.3]), вы должны записать в блок остальную информацию. Отметим, что FCB для работы с простым последовательным файлом с длиной записи, равной 128 байтам, не требует дальнейших приготовлений. После создания FCB при последующих операциях нужно, чтобы DS:DX указывали на него. Простейшая его форма такова:

```
FCB             DB  1,'FILENAMEEXT',25 DUP(0)
```

Можно также создать FCB в виде структуры:

```
FCB             STRUC
DRIVE_NUM       DB  0
FILE_NAME       DB  8 DUP(?)
FILE_EXT        DB  3 DUP(?)
BLOCK_NUM       DW  0
RECORD_SIZE     DW  0
FILE_SIZE       DD  0
FILE_DATE       DW  0
RESERVED        DB  10 DUP(0)
CURRENT_REC     DB  0
RANDOM_REC      DD  0
FCB             ENDS
```

При таком подходе программе проще помещать данные в FCB, поскольку метки существуют для каждого поля. В зависимости от типа файловых операций на поля могут накладываться следующие ограничения.

1. Для файлов прямого доступа вы должны установить размер и номер записи в поле записи прямого доступа.

2. Для доступа к началу последовательного файла необходимо установить только размер записи при условии, что вы инициализировали поля текущего блока и текущей записи в 0 (просто обнулите весь FCB, за исключением имен накопителя и файла). При открытии поле размера записи будет установлено равным 128, если это значение вас устраивает, дальнейшая подготовка не нужна.

3. Для доступа к середине или к концу последовательного файла нужно установить поля текущего блока и текущей записи (в этом случае ваша программа должна будет производить вычисления сама).

Префикс программного сегмента (см. [1.3.0]) имеет достаточно большое поле, чтобы содержать управляющий блок файла. Это пространство предоставляется для каждой программы, поэтому рекомендуется его использование для экономии памяти, особенно в программах типа .COM. Поле FCB расположено со смещением 5CH в префиксе программного сегмента. В программах COM используйте ORG для создания FCB следующим образом (здесь помечен также принятый по умолчанию DTA, который будет обсуждаться ниже):

```
;---в начале кодового сегмента
                ORG     5CH
FCB             LABEL   BYTE
DRIVE_NUM       DB      0
FILE_NAME       DB      8 DUP(?)
FILE_EXT        DB      3 DUP(?)
BLOCK_NUM       DW      0
RECORD_SIZE     DW      0
FILE_SIZE       DD      0
FILE_DATE       DW      0
RESERVED        DB      10 DUP(0)
CURRENT_REC     DB      0
RANDOM_REC      DD      0
                ORG     80H
DTA             LABEL   BYTE
                ORG     100H
                ASSUME  CS:CSEG, DS:DSEG, SS:SSEG
...
```

Расширенный FCB необходим для создания или доступа к файлу, имеющему специальные атрибуты, например к спрятанному файлу или файлу только для чтения. Различные атрибуты объяснены в [5.2.6]. Расширенный FCB на 7 байт длиннее, причем эти 7 байт предшествуют обычному блоку. Первый байт равен FF, что указывает на специальный статус. За ним следуют 5 байт ASCII 0, а затем - байт атрибутов. При открытии файла с помощью расширенного FCB DS:DX должны указывать на первый из дополнительных семи байтов, а не на имя накопителя, как для

обычного FCB. Вот общая форма, где 2 - значение байта атрибутов, а 1 - указывает на накопитель:

```
FCB   DB   0FFH, 5 DUP(0),2,1,'FILENAMEEXT',25 DUP(0)
```

Метод дескриптора файла. Этот метод требует меньшей подготовки, чем метод FCB. Для него вы должны только создать строку, указывающую путь к файлу, такую, как в стандартных командах DOS. Например, B:COMPILE\UTILITY\PASCAL указывает на файл PASCAL в подкаталоге UTILITY. Строка ограничена длиной в 63 символа, включая имя накопителя. При открытии файла (с использованием функции 3DH прерывания 21H - см. [5.3.3]), DS:DX должны указывать на первый байт этой строки. Система выполняет всю работу по анализу строки и нахождению файла, а после того как файл открыт, она возвращает 16-битовый идентификационный номер файла в AX. Его называют номером файла, и он используется во всех последующих операциях с этим файлом.

Буфера данных. Программа должна указать место в памяти, куда помещаются принимаемые данные или откуда должны браться выводимые. Это пространство в памяти может быть временным буфером, который будет использоваться данными как промежуточная станция. Или это пространство может быть именно тем местом, где данные реально обрабатываются. Обычно временный буфер устанавливается размером в одну запись и бывает удобно описать его как строковую переменную в сегменте данных, что и сделано в приведенном ниже примере. С другой стороны, большие рабочие области данных должны распределяться с помощью методов распределения памяти, предоставляемых операционной системой (см. [1.3.1]). Ведь создание, например, области данных размером в 10000 байт в сегменте данных сделает программу на диске на 10000 байт длиннее, что совершенно не нужно.

Буфер, используемый методом FCB доступа к файлам, называется областью обмена с диском или DTA. Имеется словный указатель на этот буфер, который хранится операционной системой и может быть изменен вашей программой. В фирменной документации сам указатель на DTA часто называют DTA. Поскольку указано только начало буфера, то ничто не мешает данным занять область, прилегающую к DTA, поэтому вы сами должны следить, чтобы этого не произошло. Указатель на DTA устанавливается специальной функцией DOS и после того, как он установлен, все функции чтения/записи автоматически обращаются к нему. Это означает, что сами функции не должны содержать адрес временного буфера.

Когда DTA совпадает с областью данных, в которой они обрабатываются, необходимо постоянно менять DTA, с тем чтобы файловые операции могли получать доступ к различным фрагментам данных. При простой операции последовательного чтения или при операции чтения одного блока с прямым доступом система автоматически помещает в DTA одну запись за другой. Необходимо отвести пространство, достаточное для числа записей, которые будут затребованы программой. DTA не может иметь размеры больше одного сегмента (64K).

Для установки указателя на DTA используйте функцию 1AH прерывания 21H. DS:DX должны указывать на первый байт DTA. Затем надо выполнить функцию. Это все, что вам нужно. К примеру:

```
;---в сегменте данных
DTA DB   256 DUP (?)
;---установка DTA
     LEA  DX,DTA      ;DS:DX указывают на DTA
     MOV  AH,1AH      ;функция установки DTA
     INT  21H         ;установка DTA
```

Функция 2FH прерывания 21H сообщает текущую установку указателя DTA. У нее нет входных регистров. При возврате ES:BX содержат сегмент и смещение DTA.

Префикс программного сегмента (см. [1.3.0]) обеспечивает каждую программу 128-байтовым встроенным DTA, начиная со смещения 80H и до 9FH. Вы можете применять его при нехватке памяти. Первоначально указатель на DTA установлен именно на этот буфер, поэтому, если вы будете использовать его, нет нужды устанавливать указатель. Этот буфер по умолчанию особенно удобен при работе с COM-файлами, где DS указывает на начало префикса программного сегмента. Для файлов EXE может потребоваться небольшой добавочный код, чтобы использовать DTA по умолчанию. Отметим, что определить текущую установку указателя на DTA вам может помочь функция 2FH прерывания 21H. У нее нет входных регистров, а при выходе ES:BX указывают на DTA.

Указатель на DTA не используется при доступе к файлу методом дескриптора файла. Функции чтения или записи данных всегда содержат адрес, по которому расположен буфер данных. Только от вашего желания зависит, будут ли данные передаваться через временный буфер или непосредственно в то место, где они будут обрабатываться.

5.3.6. Анализ информации командной строки

При запуске многие программы дают пользователю возможность поместить добавочную информацию в командной строке, обычно указывающую имя файла, с которым программа будет работать. Эта информация записывается в 128-байтовую область, начинающуюся со смещения 80H в префиксе программного сегмента [1.3.0]. (Эта же область используется как DTA по умолчанию, см. в [5.3.5].) Первый байт содержит длину строки, а за ним следует собственно строка.

Для работы с файлами методом дескриптора файла имя файла, вводимое в командной строке, должно иметь адекватную форму. Требуется, чтобы пользователь программы применял стандартный протокол MS-DOS для строки пути. С другой стороны, управляющий блок файла требует, чтобы строка вида 'A:ACCT.BAK' была преобразована к виду 1,'ACCT BAK'. MS-DOS имеет специальную функцию, которая выполняет такое преобразование над первой порцией информации, следующей за именем программы в командной строке. Эта процедура называется *разбором строки* (parsing).

Средний уровень

Имя файла - это первая информация, следующая за именем загружаемой программы. Оно должно быть отделено от имени программы одним из следующих символов : . ; , = + табуляцией или пробелом. Конец имени файла должен быть обозначен одним из символов : . ; , = + \ < > | / " [ ] табуляцией, пробелом или одним из управляющих символов (коды ASCII от 1 до 31).

Функция 29H прерывания 21H производит разбор имени файла. DS:SI должны указывать на смещение 81H в PSP. Помните, что при загрузке программы как DS, так и ES установлены на начало PSP. ES:DI должны указывать на область памяти, которая будет служить управляющим блоком для нового файла. Установка битов в AL определяет, как будет выполняться разборка. Имеют значение только биты 0-3:

бит	0	1 = начальный ограничитель игнорируется
	1	1 = байт, идентифицирующий накопитель, устанавливается в FCB, только если он указан в командной строке
	2	1 = имя файла в FCB меняется, только если командная строка содержит имя файла
	3	1 = расширение файла в FCB меняется, только если командная строка содержит расширение файла

После того как эта информация установлена, программа может вызывать функцию. Если в командной строке не указан накопитель, то берется накопитель по умолчанию. Если отсутствует расширение файла, то предполагается, что оно состоит из пробелов (ASCII 32). Если в имени файла указана звездочка, то она заменяется на нужное число вопросительных знаков в поле имени файла FCB. AL возвращает 1, если имя файла содержит * или ?, и FF, если указан неверный накопитель.

При возврате DS:SI указывают на первый символ, следующий за именем файла, которое начинается со смещения 81H. Дальнейшая информация, содержащаяся в командной строке, должна расшифровываться программой. ES:DI указывают на первый байт вновь сформированного FCB. Если в FCB не создано допустимого имени файла, то содержимое ES:[DI]+1 равно пробелу. В предлагаемом примере код помещается в область FCB в PSP, начиная со смещения 5CH:

```
;---разбираем командную строку, создавая FCB со смещением 5CH
;---в PSP
MOV  AH,29H             ;номер функции
MOV  SI,81H             ;DS:SI указывают на имя файла
MOV  DI,5CH             ;ES:DI указывают на область FCB
MOV  AL,1111B           ;кодовый байт
INT  21H                ;установка FCB
MOV  AL,ES:[DI] + 1     ;получение статусной информации
CMP  AL,32              ;был ли установлен файл
JE   ERROR_ROUTINE      ;если нет, то уход на обработку ошибки
```

5.4. Чтение и запись файла

Имеются два основных метода доступа к файлу - последовательный и прямой. Хотя в литературе по вычислительной технике часто используют термины "последовательный" файл и файл "прямого доступа", сами по себе файлы хранятся на диске одинаково: как непрерывная последовательность байтов. Ни в каталоге, ни в каком-либо другом месте нет индикатора, указывающего, что данный файл является последовательным или файлом прямого доступа. Реально эти два типа файлов различаются по расположению данных в них и по методу доступа к ним. К любому файлу прямого доступа можно получить последова-

тельный доступ, а к любому последовательному файлу - прямой доступ, хотя причины делать это появляются редко, особенно во втором случае.

Последовательные файлы помещают элементы данных один за другим, независимо от их длины, разделяя эти элементы парой символов: сначала возвратом каретки (ASCII 13), а затем переводом строки (ASCII 10). Языки высокого уровня, такие, как Бейсик, вставляют эти символы автоматически, в то время как программы на ассемблере должны сами заботиться о вставке этих символов после записи каждой переменной в файл. В последовательных файлах могут храниться как числа, так и строки. Строки требуют по одному байту на каждый символ строки. Числа по соглашению записываются в строковом виде, хотя есть возможность их записи и в числовом виде. Таким образом Бейсик записывает значение "128" в виде строки из трех цифр, хотя программа на ассемблере может записать это число в виде двухбайтового целого или даже однобайтового кода - все определяется тем, что при повторном чтении файла программа должна понимать используемый формат. Для совместимости рекомендуется записывать числа в виде строк.

Необязательно, чтобы каждое число строки было отделено парой возврат каретки / перевод строки, однако если эта пара опущена, то программа должна обеспечить способ отделения данных. Например, 10 целых чисел могут быть записаны как 20-байтовый элемент данных. С другой стороны, очень большие элементы данных, такие, как параграфы текста, могут быть разделены на несколько элементов данных (стандартный текстовый файл представляет из себя документ, разбитый на строки удобного размера, записанные последовательно). Поскольку элементы данных имеют переменную длину, то невозможно узнать, где в файле расположен определенный элемент. Поэтому для того чтобы найти нужный элемент, программа должна читать файл, начиная сначала и отсчитывая нужное число пар возврат каретки / перевод строки. По этой причине файлы такого формата называют последовательными. Как правило, с диска в память передается весь такой файл.

Файлы прямого доступа заранее отводят фиксированное место под каждый элемент данных. Если какой-то элемент данных не занимает все отведенное пространство, то остаток заполняется пробелами. Если каждый элемент занимает 10 байт, то легко можно просмотреть сразу 50-й элемент, поскольку можно вычислить, что он начинается с 491-го байта файла (т.е. с байта #490, поскольку отсчет начинается с 0). Как правило, связанный набор элементов группируется в запись. Каждая запись содержит несколько полей, которые создают набор номеров байтов, начиная с которых пишутся данные элементы. Например, запись может иметь поля:

возраст, вес и рост. Соответственно эти поля могут занимать 2, 3 и 5 байт, а вместе образовывать запись длиной в 10 байт. Файл прямого доступа может состоять из тысяч таких записей. Каждая запись следует непосредственно за предшествующей без всяких ограничителей, таких, как пары возврат каретки/перевод строки, используемые в последовательных файлах. При этом записи можно делать в любом порядке и можно вывести запись 74, хотя запись 73 еще не была выведена (при этом для записи 73 отведено дисковое пространство, и она будет содержать те данные, которые случайно оказались в том секторе, в котором отведено место для этой записи). В отличие от последовательных файлы прямого доступа остаются на диске. В памяти присутствуют только определенные записи, с которыми идет работа в данный момент времени.

Когда для прямого доступа к файлу используется управляющий блок файла, системе сообщается размер записи файла (все записи данного файла должны иметь одинаковую длину). Это позволяет программе запросить любую запись по номеру, а MS-DOS точно вычислит, где эта запись расположена на диске. При работе с файлами прямого доступа методом дескриптора файла программа должна сама вычислять положение требуемой записи.

Система хранит файловый указатель для каждого буфера файла. Он указывает на n-й байт файла, определяя место в файле, с которого будет начинаться следующая операция чтения или записи. При последовательной операции перезаписи файловый указатель первоначально устанавливается на начало файла и постоянно сдвигается по мере того, как все новые и новые данные записываются в файл. Когда данные добавляются к последовательному файлу, то файловый указатель первоначально устанавливается на конец файла. При обращении к одной записи в файле прямого доступа положение записи вычисляется в виде смещения относительно начала файла и указатель устанавливается равным этому значению; затем нужная запись читается или пишется. Обычно за файловым указателем следит система, однако программа может сама управлять им и манипулировать для своих специальных нужд.

Единственным примером низкого уровня в данном разделе является чтение/запись одного сектора. Чтение или запись целых файлов состоит в последовательности таких чтений или записей одного сектора, программируя микросхему контроллера НГМД заново для каждого сектора. Полномасштабные файловые операции очень сложны на этом уровне, что следует хотя бы из больших размеров файла COMMAND.COM. Однако, изучив обсуждение операций низкого уровня, а также имея информацию о таб-

лице размещения файлов [5.1.1] и дисковых каталогах [5.2.1], вы можете представить, как работает дисковая операционная система.

5.4.1. Программирование контроллера НГМД 765 и микросхемы прямого доступа к памяти 8237

Микросхема контроллера НГМД 765 фирмы NEC управляет мотором и головками накопителя на дискетах и обрабатывает потоки данных, направляемые в или из дисковых секторов. Один контроллер, установленный на плате адаптера дисков, может обслуживать до четырех НГМД. За исключением случаев, связанных с защитой от копирования, программировать микросхему контроллера НГМД прямо не приходится. Процедуры работы с дисками, предоставляемые DOS и BIOS, эффективны и удобны, и, кроме того, очень рискованно писать свои собственные процедуры, поскольку ошибки в них могут разрушить дисковый каталог или таблицу размещения файлов, что вызовет полное разрушение информации на диске.

Предлагаемое в этом параграфе обсуждение вопросов ставит перед собой цель дать вам только общее представление о них. Листинг ROM-BIOS, приведенный в конце каждого технического руководства по MS-DOS, содержит код тщательно разработанных процедур для форматирования дискет, чтения и записи секторов, а также сброса и получения статуса накопителей. После того как вы усвоите приведенный здесь материал, изучите процедуры ROM-BIOS для продолжения вашего образования в области операций с дисками на низком уровне. Вам потребуется также документация по микросхеме контроллера НГМД 8272A фирмы Intel, которая аналогична микросхеме фирмы NEC. В данной документации перечислены прерывания, генерируемые контроллером НГМД, в то время как в документации по IBM PC этого списка нет. Информация о микросхеме 8272A может быть найдена во втором томе справочника по компонентам микросистем (Microsystem Components Handbook, Volume 2).

Контроллер НГМД может выполнять 15 операций, из которых здесь будут обсуждаться только три: поиск, чтение или запись одного сектора. Понимание работы этих операций позволит вам выполнить любую из оставшихся двенадцати при условии, что у вас будет упомянутая выше информация. Чтение файла состоит в поиске его в каталоге [5.2.1], определении его положения на диске с помощью таблицы размещения файлов [5.1.1] и затем наборе операций чтения одного сектора. Эта процедура включает 6 шагов.

1. Включение мотора и короткое ожидание, пока он наберет обороты.

2. Выполнение операции поиска и ожидание прерывания, указывающего на завершение этой операции.

3. Инициализация микросхемы DMA для пересылки данных в память.

4. Посылка команды чтения контроллеру НГМД и ожидание прерывания, указывающего, что пересылка данных завершена.

5. Получение информации о статусе контроллера НГМД.

6. Выключение мотора.

Контроллер НГМД работает через три порта ввода/вывода (на самом деле микросхема имеет больше чем три регистра, но доступ к большинству из них осуществляется через один порт):

3F2H	регистр цифрового вывода
3F4H	регистр статуса
3F5H	регистр данных

Первый шаг состоит в доступе к регистру цифрового вывода. Значение его битов следующее:

биты 1-0	выбор накопителя, где	00 = A
		01 = B
		10 = C
		11 = D
2	0 = сброс контроллера НГМД	
3	1 = разрешение прерывания FDC и доступа DMA	
7-4	1 = включение мотора накопителя D-A (бит 4 = A)	

Это регистр только для записи, поэтому необходимо заботиться обо всех его битах. В приведенном ниже примере используется накопитель A, поэтому цепочка битов должна иметь вид: 00011100. Такая установка битов выбирает накопитель A, сохраняет установленным бит 2, разрешающий работу с НГМД, и включает мотор накопителя A. Не сбрасывайте бит 2 в нуль, так как в этом случае вам придется производить перекалибровку накопителя - действие, которое очень редко бывает необходимо.

"Перекалибровка" накопителя подразумевает возврат его головки на нулевую дорожку. Эта операция осуществляется посылкой простой последовательности команд контроллеру НГМД. Контроллер НГМД управляет текущей позицией головки посредством запоминания всех изменений позиции головки после ее начальной установки на нулевую дорожку. Когда контроллер НГМД сбрасывается за счет изменения бита 2 регистра цифрового вывода, то значение текущей позиции головки устанавливается в нуль, неза-

висимо от того, на какой дорожке находится головка на самом деле, что делает необходимым перекалибровку. Обычно сброс контроллера НГМД производится только в случае такой серьезной ошибки накопителя, после которой неизвестно текущее состояние контроллера НГМД и накопителя.

Отметим, что выбор накопителя и включение его мотора - отдельные действия. Контроллер НГМД может иметь доступ только к одному накопителю в данный момент времени, но моторы могут быть включены у нескольких. Моторы могут оставаться включенными еще несколько секунд после завершения обмена данными в ожидании следующего доступа к накопителю. Такая стратегия позволяет избежать потери времени на повторное ожидание, пока мотор наберет скорость. Напротив, мотор нельзя оставлять постоянно включенным, так как это приведет к преждевременному износу дискет.

Работа микросхемы контроллера НГМД делится на три фазы: командную, выполнения и результата. В командной фазе один или более байтов посылаются в регистр данных. Последовательность байтов строго фиксирована и меняется от команды к команде. Затем контроллер НГМД выполняет команду, и в это время он находится в фазе выполнения. Наконец, во время фазы результата ряд байтов статуса считываются из регистра данных. При этом необходимо, чтобы не было ошибки в числе передаваемых или считываемых данных в регистр данных в фазах командной и результата.

Число байтов команды и результата меняется в зависимости от выполняемой контроллером дисковой операции. В техническом руководстве по IBM PC приведены данные для всех 15 операций. Первый байт команды является кодом, определяющим требуемую операцию. Номер кода содержится в младших пяти битах байта, в некоторых случаях в старших трех битах закодирована добавочная информация. Часто второй байт команды содержит номер накопителя (0-3) в младших двух битах и номер головки (0 или 1) в бите 2, все остальные биты контроллером НГМД игнорируются. При операции поиска требуется дополнительно еще только один байт, в котором должен содержаться номер новой дорожки. Для чтения или записи сектора требуется семь дополнительных командных байтов, которые идентичны в этих двух случаях. Байты с третьего по пятый содержат текущий номер дорожки, номер головки и номер сектора. За ними следуют четыре байта, содержащие техническую информацию, необходимую для контроллера НГМД.

Первый байт этой технической информации относится к числу байтов в секторе, которое кодируется как 0 для 128, 1 для 256, 2 для 512 и 3 для 1024. Конечно, дискеты, созданные в MS-DOS, имеют сектора размером 512 байт. Затем идут данные конца

дорожки (EOT), которые дают максимальный номер сектора для цилиндра; это значение равно 9 для дискет емкостью 360K. Наконец, следует байт, представляющий длину сдвига (GPL, равный 2AH) и длину данных (DTL, равный FFH). Техническое руководство по IBM PC содержит таблицу, где объясняются другие входные параметры, например те, которые используются при форматировании диска. MS-DOS хранит четыре технических параметра в памяти, в специальной таблице параметров, называемой *базой диска* (disk base). Вектор прерывания 1EH указывает на эту таблицу. Значения параметров хранятся в том порядке, в котором они должны быть переданы контроллеру НГМД, начиная со смещения 3. В следующей таблице показана командная последовательность для трех операций, используемых в приведенном ниже примере. В цепочках битов черех X обозначены биты, значение которых несущественно, через H - номер головки, а через DD - номер накопителя.

Операция	# байта	Функция	Установка для головки 0 дорожки 15, сектора 1
Поиск	1	номер кода 00001111	1FH
	2	головка и накопитель XXXXXHDD	00H
Чтение сектора	1	номер кода01100110	66H
	2	головка и накопитель XXXXXHDD	00H
	3	номер дорожки	0FH
	4	номер головки	00H
	5	номер сектора	01H
	6	байтов в секторе	02H
	7	конец дорожки	09H
	8	длина сдвига	1AH
	9	длина данных	FFH
Запись сектора	1	номер кода 01000101	45H
	2-9	те же, что и для чтения сектора	

Вы должны быть уверены, что контроллер НГМД готов, прежде чем вы пошлете или прочитасте байт из регистра данных.

Биты 7 и 6 регистра статуса предоставляют эту информацию. Ниже перечислены значения битов этого регистра:

биты	3-0	1 = накопитель D-A в режиме поиска
	4	1 = контроллер НГМД выполняет команду чтения/записи
	5	1 = контроллер НГМД не в режиме DMA
	6	1 = регистр данных контроллера НГМД готов к приему данных
		0 = готов к посылке данных
	7	1 = контроллер НГМД готов к посылке или приему данных

Перед началом операций с диском неплохо проверить, равен ли бит 6 нулю, индицируя, что контроллер НГМД ожидает команду. Если он ожидает посылки данных, то произошла ошибка. Когда байт данных посылается в регистр данных, то бит 7 регистра статуса становится равным нулю; продолжайте чтение регистра до тех пор, пока бит не изменится обратно на 1, а затем посылайте следующий байт команды. Аналогично проверяйте бит статуса перед чтением байта статуса в фазе результата. В конце приведенного ниже примера показаны две процедуры, которые выполняют эти функции.

Когда операция поиска завершена, то контроллер НГМД инициирует прерывание 6, прерывание от НГМД. Хотя так же просто можно узнать об окончании операции поиска, проверяя регистр статуса, в примере это делается за счет обработки прерывания. Когда происходит прерывание, то обработчик прерывания BIOS устанавливает бит 7 байта статуса поиска в области данных BIOS, расположенного по адресу 0040:003E. Это единственный результат обработки прерывания. Можно проверять этот байт до тех пор, пока бит 7 не будет установлен, а затем переходить к следующему шагу операции чтения сектора.

Следующий шаг состоит в инициализации микросхемы прямого доступа к памяти 8237. Она предназначена для обмена данными между периферийными устройствами и памятью, т.е. для работы, которой может заниматься также процессор. На самом деле в PCjr, где нет микросхемы DMA, контроллер НГМД посылает данные прямо в процессор, который в свою очередь пересылает их в память. Тактовая частота процессора адекватна этой задаче, однако при пересылке данных все прерывания должны быть запрещены, с тем чтобы не происходило потери данных. Это означает, что в PCjr при передаче данных ввод с клавиатуры или из модема запрещен. Прерывания таймера также игнорируются, однако впос-

ледствии счетчик времени суток обновляется специальной процедурой, использующей канал 1 микросхемы таймера 8253 для подсчета импульсов, прошедших за время дисковых операций. Все остальные модели IBM PC имеют микросхему DMA, поэтому процессор свободен при передаче данных.

В IBM PC и XT применяется четырехканальная микросхема DMA 8237. Канал 0 предназначен для "освежения" памяти (memory refresh); он постоянно восстанавливает заряд ячеек оперативной памяти. Если вы будете работать по этому каналу, то это приведет скорее всего к поломке машины. Канал 2 предназначен для дисковых операций, а два других канала, с номерами 1 и 3, доступны (через разъемы расширения) для дополнительного оборудования. К сожалению, обмен память-память требует двух каналов и одним из них должен быть канал 0, поэтому такой обмен недоступен на IBM PC и XT. Однако AT имеет 7 каналов прямого доступа к памяти и DMA автоматически используется инструкциями MOVS, существенно увеличивая производительность.

Перед инициализацией канала программа должна послать в микросхему код, сообщающий, будет ли происходить чтение или запись в контроллер НГМД. Этот однобайтовый код равен 46H для чтения и 4AH для записи. Он должен быть послан в каждый из двух портов с адресами 0BH и 0CH.

Каждый канал микросхемы 8237 использует три регистра. Один 16-битовый регистр, регистр счетчика, содержит число передаваемых байтов данных. Его величина должна быть на единицу меньше, чем требуемое число байтов. Для канала 2 доступ к этому регистру осуществляется через порт 05H; пошлите в него два последовательных байта, причем сначала младший байт.

Остальные два регистра содержат адрес буфера в памяти, с которым будет происходить обмен данными. Этот адрес задается как 20-битовое число, поэтому, например, адрес 3000:ABCD задается как 3ABCD. Младшие 16 бит посылаются в регистр адреса, который для канала 2 имеет адрес порта 04H. Сначала посылается младший байт. Старшие 4 бита идут в регистр страницы, который для канала 2 имеет адрес порта 81H. Когда байт посылается по этому адресу, то значение имеют только 4 младших бита. Если буфер создается в сегменте данных, то вам нужно сложить значение DS и смещение буфера для получения 20-битового значения.

Сложение может привести к переносу в значении регистра страницы. Например, если DS равен 1F00H, а смещение буфера - 2000H, то результирующий адрес будет равен: 1F00 + 2000 = = 21000H.

После того как эти три регистра установлены, пошлите 2 в порт с адресом 0AH, чтобы разрешить канал 2. Это оставляет микросхему DMA в состоянии ожидания данных от накопителя, а программа должна немедленно начать посылку командных байтов в контроллер НГМД. Вот краткий перечень шагов при программировании микросхемы 8237.

1. Послать код чтения или записи.
2. Вычислить 20-битовый адрес памяти буфера, в который будут посланы данные, и заслать его в регистры адреса и страницы канала 2.
3. Поместить значение числа передаваемых байтов (минус 1) в регистр счетчика канала 2.
4. Разрешить канал.

После посылки командных байтов снова ожидайте прерывания и обращайтесь с ним так же, как и после операции поиска. Затем прочитайте байты статуса. Они таковы:

Операция	# байта	Функция
Поиск	нет	
Чтение	1	байт статуса 0
	2	байт статуса 1
	3	байт статуса 2
	4	номер дорожки
	5	номер головки
	6	номер сектора
	7	код байтов на сектор (0-3)
Запись	1-7	то же, что и для чтения

Вот значения битов трех байтов статуса:

Байт статуса 1:

биты 7-6 00 = нормальное завершение
01 = начато выполнение, не может завершиться
10 = неверная команда
11 = не выполнено, так как накопитель не подключен

	5	1 = выполняется операция поиска
	4	1 = ошибка накопителя
	3	1 = накопитель не готов
	2	номер выбранной головки
	1-0	номер выбранного накопителя

Байт статуса 2:

биты	7	1 = номер затребованного сектора больше максимума
	6	не используется (всегда 0)
	5	1 = ошибка передачи данных
	4	1 = переполнение данных
	3	не используется (всегда 0)
	2	1 = не может найти или прочитать сектор
	1	1 = не может записать из-за защиты от записи
	0	1 = отсутствует адресная метка при форматизации

Байт статуса 3:

биты	7	не используется (всегда 0)
	6	1 = встречена адресная метка удаленных данных
	5	1 = ошибка циклического контроля четности данных
	4	1 = проблема с идентификацией дорожки
	3	1 = условие команды сканирования удовлетворено
	2	1 = условие команды сканирования не удовлетворено
	1	1 = плохая дорожка
	0	1 = отсутствует адресная метка

Как видите, большая часть информации относится к форматированию диска, которое нас в настоящий момент не интересует. Однако имеется еще четвертый байт статуса, который содержит полезную информацию:

Байт статуса 3:

биты	7	1 = ошибка накопителя
	6	1 = диск защищен от записи
	5	1 = накопитель готов
	4	1 = текущая позиция головки известна
	3	1 = дискета двухсторонняя
	2	номер выбранной головки
	1-0	номер выбранного накопителя

Вы можете получить четвертый байт статуса, послав контроллеру НГМД команду "Определи статус накопителя" (Sense Drive

Status). Первый байт этой двухбайтовой команды - это число 4, а второй байт содержит номер накопителя в битах 1 и 0 и номер головки в бите 2. Единственным результатом этой операции является байт статуса 3. Отметим, что после каждой дисковой операции, если вы используете процедуры DOS или BIOS, результирующие байты статуса помещаются в область данных BIOS, начиная с адреса 0040:0042. Операционная система хранит также байт статуса дискеты по адресу 0040:0041, значение битов которого следующее:

Значение бита	Ошибка
80H	нет ответа на присоединение накопителя
40H	операция поиска неуспешна
20H	ошибка контроллера НГМД
10H	ошибка данных при чтении (ошибка CRC)
09H	попытка прямого доступа за границу 64K
08H	переполнение DMA
04H	затребованный сектор не найден
02H	не найдена адресная марка
01H	послана неверная команда контроллеру НГМД

В заключение приводим полную процедуру чтения диска, которая читает один сектор данных с дорожки 12, сектора 1, стороны 0 накопителя A в 512-байтовый буфер в сегменте данных. Семь байтов статуса также считываются в отведенный буфер. Эта процедура предназначена для IBM PC и XT. Вам необходимо воспользоваться техническим руководством по PCjr или AT, если вы работаете на этих машинах. На AT надо изменить циклы задержки, чтобы учесть большую скорость процессора, и не забывать добавлять оператор JMP SHORT $+2 между последовательными командами OUT, относящимися к одному и тому же порту. Работа с фиксированным диском осуществляется аналогично, поэтому вы можете применить изученные вами концепции в других ситуациях.

```
;---в сегменте данных
BUF      DB     512 DUP(?)
STA_BUFDB       7 DUP(?)
S_READ PROC                    ;начало процедуры чтения одного сектора
;---включение мотора
         STI                   ;прерывания должны быть разрешены
         MOV    DX,3F2H        ;адрес регистра цифрового вывода
```

```
        MOV   AL,28             ;устанавливаем биты 2, 3 и 4
        OUT   DX,AL             ;посылаем команду
;---ожидаем, пока мотор наберет скорость (около 1/2 секунды)
        MOV   CX,3500           ;счетчик цикла задержки (для IBM PC и XT)
MOT_D:  LOOP  MOT_D             ;ожидаем 1/2 секунды
;---выполняем операцию поиска
        MOV   AH,15             ;номер кода
        CALL  OU_FDC            ;посылаем контроллеру НГМД
        MOV   AH,0              ;номер накопителя
        CALL  OU_FDC            ;посылаем контроллеру НГМД
        MOV   AH,12             ;номер дорожки
        CALL  OU_FDC            ;посылаем контроллеру НГМД
        CALL  WA_INT            ;ожидаем прерывания от НГМД
;---ожидаем установки головки (25 мс)
        MOV   CX,1750           ;счетчик цикла задержки (для IBM PC и XT)
WA_SET: LOOP  WA_SET            ;ожидаем 25 мс
;---начинаем инициализацию микросхемы DMA
        MOV   AL,46H            ;код чтения данных контроллера НГМД
        OUT   12,AL             ;посылаем код по двум адресам
        OUT   11,AL
;---вычисляем адрес буфера
        MOV   AX,OFFSET BUF;берем смещение буфера в DS
        MOV   BX,DS             ;помещаем DS в BX
        MOV   CL,4              ;готовим вращение старшего нибла
        ROL   BX,CL             ;вращаем младшие 4 бита
        MOV   DL,BL             ;копируем DL в BL
        AND   DL,0FH            ;чистим старший нибл в DL
        AND   BL,0F0H           ;чистим младший нибл в BX
        ADD   AX,BX             ;складываем
        JNC   NO_CRY            ;если не было переноса, то # страницы в DL
        INC   DL                ;увеличиваем DL, если был перенос
NO_CRY: OUT   4,AL              ;посылаем младший байт адреса
        MOV   AL,AH             ;сдвигаем старший байт
        OUT   4,AL              ;посылаем младший байт адреса
        MOV   AL,DL             ;засылаем номер страницы
        OUT   81H,AL            ;посылаем номер страницы
;---конец инициализации
        MOV   AX,511            ;значение счетчика
        OUT   5,AL              ;посылаем младший байт
```

```
        MOV   AL,AH           ;готовим старший байт
        OUT   5,AL            ;посылаем старший байт
        MOV   AL,2            ;готовим разрешение канала 2
        OUT   10,AL           ;DMA ожидает данные
;---получаем указатель на базу диска
        MOV   AL,1EH          ;номер вектора, указывающего на таблицу
        MOV   AH,35H          ;номер функции
        INT   21H             ;выполняем функцию
;---посылаем параметры чтения
        MOV   AH,66H          ;код чтения одного сектора
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        MOV   AH,0            ;номера головки и накопителя
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        MOV   AH,12           ;номер дорожки
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        MOV   AH,0            ;номер головки
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        MOV   AH,1            ;номер записи
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        MOV   AH,ES:[BX] + 3  ;код размера сектора
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        MOV   AH,ES:[BX] + 4  ;номер конца дорожки
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        MOV   AH,ES:[BX] + 5  ;длина сдвига
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        MOV   AH,ES:[BX] + 6  ;длина данных
        CALL  OU_FDC          ;посылаем контроллеру НГМД
        CALL  WA_INT          ;ожидаем прерывание от НГМД
;---читаем результирующие байты
        MOV   CX,7            ;берем 7 байт статуса
        LEA   BX,STA_BUF      ;помещаем в буфер статуса
NEXT:   CALL  IN_FDC          ;получаем байт
        MOV   [BX],AL         ;помещаем в буфер
        INC   BX              ;указываем на следующий байт буфера
        LOOP  NEXT            ;повторяем операцию
;---выключение мотора
        MOV   DX,3F2H         ;адрес регистра цифрового вывода
        MOV   AL,12           ;оставляем биты 3 и 4
        OUT   DX,AL           ;посылаем новую установку
```

```
        RET                     ;конец процедуры
S_READ  ENDP
WA_INT  PROC                    ;ожидание прерывания от НГМД
;---управление статусом прерывания 6 в байте статуса BIOS
        MOV   AX,40H            ;сегмент области данных BIOS
        MOV   ES,AX             ;помещаем в ES
        MOV   BX,3EH            ;смещение для байта статуса
AGAIN:  MOV   DL,ES:[BX]        ;получаем байт
        TEST  DL,80H            ;проверяем бит 7
        JZ    AGAIN             ;до тех пор, пока не установлен
        AND   DL,01111111B      ;сбрасываем бит 7
        MOV   ES:[BX],DL        ;заменяем байт статуса
        RET
WA_INT  ENDP
OU_FDC  PROC ;посылаем байт из AH FDC
        MOV   DX,3F4H           ;адрес порта регистра статуса
KE_TR:  IN    AL,DX             ;получаем значение
        TEST  AL,128            ;бит 7 установлен?
        JZ    KE_TR             ;если нет, то снова проверяем
        INC   DX                ;указываем на регистр данных
        MOV   AL,AH             ;передаваемое значение в AH
        OUT   DX,AL             ;посылаем значение
        RET
OU_FDC  ENDP
IN_FDC  PROC                    ;получаем байт от FDC в AL
        MOV   DX,3F4H           ;адрес порта регистра статуса
ON_AG:  IN    AL,DX             ;получаем значение
        TEST  AL,128            ;бит 7 установлен?
        JZ    ON_AG             ;если нет, то проверяем снова
        INC   DX                ;указываем на регистр данных
        IN    AL,DX             ;читаем байт из регистра данных
        RET
IN_FDC  ENDP
```

5.4.2. Чтение/запись определенных секторов

Чтение или запись определенных секторов диска в основном используется при доступе к каталогам диска или его таблице размещения файлов, сектора для которых всегда расположены в одном и том же месте. В то время как чтение секторов достаточно

безобидно, запись абсолютного сектора требует, чтобы код был тщательно проверен перед работой. Ошибка может сделать каталог или таблицу размещения файлов нечитаемыми, что эквивалентно разрушению всех данных на диске.

Как DOS, так и BIOS предоставляют функции для чтения и записи определенных секторов. Однако они указывают сектора по-разному. Для IBM PC, XT и PCjr процедура BIOS требует информацию о номерах стороны (0 или 1), дорожки (0-39) и сектора (1-8). Из-за ограничения максимального номера сектора, равного 8, этот метод практически бесполезен для перечисленных машин. Однако для AT номер сектора может меняться до 8, 9 или 15, а число дорожек - до 39 или 79. Функции DOS указывают сектор одним номером, который называется *логическим номером сектора.* Начиная с наружного обода диска, секторам присваиваются последовательно возрастающие номера. Этот метод может быть использован для дисков произвольного размера и типа.

Отсчет логических секторов начинается со стороны 0 дорожки 0 сектора 1 и продолжается на стороне 1 с дорожки 0, после чего переходит на сторону 0 дорожку 1 и т.д. (На больших фиксированных дисках сначала проходится весь внешний цилиндр.) В зависимости от того, как был форматирован диск, при переходе на следующую дорожку логический номер сектора увеличивается на определенную величину. Для дискет емкостью 360K каждая дорожка (с учетом обеих сторон) добавляет к логическому номеру 18. Однако вычисления немного усложняются тем, что отсчет начинается с нуля. Таким образом, первый сектор на дорожке 3 стороны 2 должен иметь номер, равный 3 x 18, для дорожек 0-2 плюс 9 для стороны 0 дорожки 3 плюс единица, указывающая на первый сектор дорожки 3 стороны 1. Эта сумма равна 64. Логический номер сектора на 1 меньше этого числа. На рис. 5.4 сравниваются методы указания сектора DOS и BIOS.

Высокий уровень

Бейсик не предоставляет прямого доступа к секторам диска. Надо использовать соответствующую процедуру на машинном языке. В приложении Г объясняется логика взаимодействия с этой процедурой. В примере читается 9 секторов дорожки 3 стороны 1 дискеты емкостью 360K. Сама процедура размещается в памяти, начиная с адреса сегмента &H1000, а содержимое секторов размещается, начиная с сегментного адреса &H2000 (напоминаем, что абсолютный адрес равен сегментному адресу, умноженному на 16). Для того чтобы записать на диск содержимое этого буфера, надо изменить в данных программы седьмой байт с конца &H25 на &H26. Все остальное остается неизменным.

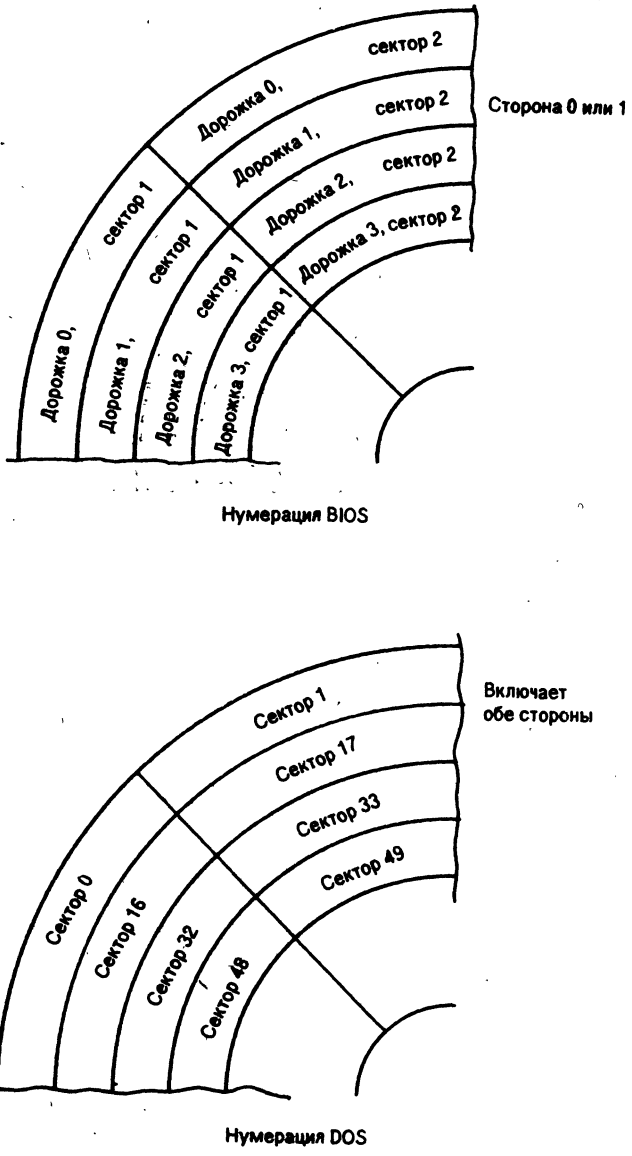


Рис. 5.4. Нумерация секторов диска, используемая BIOS и DOS

```
100 DEFINT A-Z                  'все переменные будут целыми
110 DATA &H55, &H8B, &HEC, &H1E, &H8B, &H176, &H0C, &H18B
120 DATA &H04, &H8B, &H76, &H0A, &H8B, &H14, &H18B, &H176
130 DATA &H08, &H8B, &H0C, &H8B, &H76, &H06, &H8A, &H1C
140 DATA &H8E, &HD8, &H8B, &HC3, &H8B, &H00, &H00, &HCD
150 DATA &H25, &H59, &H1F, &H5D, &HCA, &H08, &H00
160 DEF SEG = &H1000            'помещаем процедуру по адресу &H10000
170 FOR N = 0 TO 38             'для каждого байта процедуры
180 READ Q: POKE N,Q            'читаем байт и помещаем его в память
190 NEXT                        'следующий байт
200 READSECTOR = 0              'выполняем код, начиная с первого байта
210 BUFFER = &H2000             'буфер для данных имеет адрес &H120000
220 LOGICALNUMBER = 62          'логический номер сектора равен 62
230 NSECTORS = 9                'число считываемых секторов
240 DRIVE = 0                   'номер накопителя (0 = A)
250 CALL READSECTOR (BUFFER, LOGICALSECTORS, NSECTORS, DRIVE)
260 'теперь сектора в памяти, начиная с адреса 2000:0000
```

Средний уровень

BIOS использует функцию 2 прерывания 13H для чтения секторов и функцию 3 прерывания 13H для записи секторов. В обоих случаях DL должен содержать номер накопителя от 0 до 3, где 0 = A, 1 = B и т.д., DH - номер головки (стороны), 0-1. CH должен содержать номер дорожки от 0 до 39, а CL - номер сектора от 0 до 8. AL содержит число секторов, которое необходимо считать. Допускается сразу читать не более восьми секторов, что вполне достаточно для большинства целей. ES:BX должны указывать на начало буфера в памяти, куда будут помещаться данные или откуда они будут браться. При возврате AL будет содержать число прочитанных или записанных секторов. Если операция завершена успешно, то флаг переноса будет равен нулю. Если он равен 1, то AH будет содержать байт статуса дисковой операции, описанный в [5.4.8].

```
;---в сегменте данных
BUFFERDB   4000 DUP(?)          ;создаем буфер
;---читаем сектора
        MOV AX,SEG BUFFER       ;ES:BX должны указывать на буфер
        MOV ES,AX
        MOV BX,OFFSET BUFFER
        MOV DL,0                ;номер накопителя
        MOV DH,0                ;номер головки
```

```
        MOV CH,0            ;номер дорожки
        MOV CL,1            ;номер сектора
        MOV AL,1            ;число секторов для чтения
        MOV AH,2            ;номер функции чтения
        INT 13H
```

Прерывания DOS 25H и 26H читают и записывают абсолютные сектора диска соответственно. Надо поместить логический номер стартового сектора в DX, а DS:BX должны указывать на буфер. CX содержит число секторов для чтения или записи, а AL - номер накопителя, где 0 = A, 1 = B и т.д. Процедуры портят все регистры, кроме сегментных. При возврате регистр флагов остается на стеке, оставляя стек невыровненным. Не забудьте убрать это значение со стека сразу после возврата (в примере это значение убирается в CX).

```
;---в сегменте данных
BUFFER DB   DUP 5000(?)     ;создаем буфер
;---читаем сектора
        PUSH DS             ;сохраняем регистры
        MOV  AX,SEG BUFFER  ;DS:BX должны указывать на буфер
        MOV  DS,AX
        MOV  BX,OFFSET BUFFER
        MOV  DX,63          ;логический номер сектора
        MOV  CX,9           ;читаем всю дорожку
        MOV  AL,0           ;накопитель A
        INT  25H            ;функция чтения секторов
        POP  CX             ;убираем из стека флаги
        POP  DS             ;восстанавливаем регистры
        JNC  NO_ERROR       ;если нет ошибки, то на продолжение
        CMP  AH,3           ;проверка возможных ошибок

.
.
NO_ERROR:                   ;продолжение программы
```

Если при возврате флаг переноса равен 1, то произошла ошибка и в этом случае AH и AL содержат два отдельных байта статуса ошибки. Если AH = 4, то указанный сектор не найден, а если AH = 2, то диск неверно отформатирован. Если AH = 3, то была попытка записи на дискету, защищенную от записи. Все остальные значения AH говорят об аппаратной ошибке.

Низкий уровень

Дисковые операции на низком уровне требуют прямого программирования микросхем контроллера НГМД и прямого доступа к памяти. Поскольку эти операции взаимосвязаны, они рассматриваются вместе в параграфе [5.4.1].

5.4.3. Запись в последовательные файлы

С точки зрения программиста языки высокого уровня работают с последовательными файлами порциями в одну единицу данных. Один оператор "записывает" содержимое переменной в последовательный файл, ограничивая ее парой возврат каретки/перевод строки. С другой стороны, программы, написанные на языке ассемблера, имеют дело с данными, измеряемыми в единицах записей. Они помещают данные в буфер, который может содержать одну или несколько записей, добавляя пары возврат каретки/перевод строки между элементами данных, а не между записями. Некоторые элементы данных могут принадлежать двум записям. Тогда для записи используется функция MS-DOS, позволяющая записать на диск одну или несколько записей. На всех уровнях программирования DOS может не производить физической записи на диск каждый раз, когда была подана команда вывода. Вместо этого в целях экономии DOS ожидает, пока его выходной буфер будет заполнен, прежде чем записать данные на диск.

Отметим, что Бейсик автоматически добавляет в конец записываемого им последовательного файла символ с кодом ASCII 26 (Ctrl-Z). Это требование стандартных текстовых файлов. Функции DOS не добавляют этот символ; ваша программа должна сама записать его в конец элемента данных. Файлы прямого доступа не ограничиваются символом ASCII 26.

Высокий уровень

Бейсик готовит файлы к последовательной записи, открывая файл в режиме последовательного доступа оператором OPEN. Этот оператор имеет две формы и выбор одной из них целиком зависит от вашего вкуса. Форматы оператора OPEN таковы:

```
100 OPEN "MYFILE" FOR OUTPUT AS #1
```

или

```
100 OPEN "O", #1, "MYFILE"
```

Во второй форме буква "O" обозначает вывод (output). Символ #1 обозначает кодовый номер, по которому вы будете впоследствии обращаться к файлу в операторах доступа, таких, как WRITE #1 или INPUT #1. В обоих случаях открывается файл с именем MYFILE для приема данных в последовательном режиме. Если файл с таким именем не найден на диске, то оператор OPEN создаст его. Если же файл существует, то он будет перезаписан, т.е. после закрытия он будет содержать только новые записанные в него данные. Чтобы добавить данные в конец существующего последовательного файла, не изменяя его предыдущего содержимого, нужно открыть его, используя первую форму оператора OPEN в виде OPEN "MYFILE" FOR APPEND AS 1. Более подробно об этом см. в [5.3.3].

Данные записываются в файл с помощью операторов PRINT # и WRITE #. Они имеют одинаковую форму:

```
100 PRINT #1, S$
```

или

```
100 WRITE #1, X
```

#1 относится к идентификационному номеру файла (дескриптору файла), присваиваемому ему оператором OPEN. В первом примере в файл записывается значение строковой переменной, а во втором - численное значение, но можно любым из операторов записывать и то, и другое. Численные значения записываются в последовательные файлы в строковом виде, хотя они и берутся не из строковых переменных. Например, 232 является двухбайтовым целым в строковой форме, однако если X = 232, то оператор PRINT #1, X помещает в файл три байта, используя коды ASCII для цифр 2, 3 и 2.

Операторы PRINT # и WRITE # отличаются друг от друга способом отделения элементов данных в файле. Какой из них более подходящий, определяется характеристиками данных. Основное различие между двумя операторами состоит в том, что WRITE # вставляет дополнительные ограничители между элементами данных. Рассмотрим случай, когда оператор выводит несколько переменных в виде 100 PRINT #1, A$, Z, B$ или 100 WRITE #1, A$, Z, B$. В этом примере пара возврат каретки/перевод строки будет помещена в файл только за последней из трех переменных (отметим, что строковые и числовые переменные могут быть перемешаны). Как же можно впоследствии выделить эти три переменные? Если был использован оператор PRINT #, то никак. Все три переменные будут объединены в непрерывную строку. Если же

был использован оператор WRITE#, то каждый элемент данных будет заключен в кавычки, а между ними будут стоять запятые. Затем, при чтении этих элементов из файла, Бейсик будет автоматически удалять кавычки и запятые, которые были добавлены оператором WRITE#.

Кроме того, имеется еще ряд менее важных вопросов. Один из них состоит в том, что вся проблема с ограничителями может быть снята, если использовать для вывода каждой переменной отдельный оператор PRINT# или WRITE#. В этом случае PRINT# будет отделять все элементы парами возврат каретки/перевод строки, а WRITE# будет делать то же самое, но по-прежнему каждый элемент будет заключен в кавычки (что напрасно расходует файловое пространство). Более того, для вывода строк, которые сами содержат кавычки, оператор WRITE# использовать нельзя, поскольку первая же внутренняя кавычка будет при чтении ошибочно воспринята как признак конца переменной. И наконец, отметим, что когда в одном операторе выводится несколько переменных, то они оба форматируют данные в точности так же, как они форматировались бы при выводе на терминал. Таким образом, PRINT #1, A$, B$ отделяет B$ от A$, в то время как PRINT #1, A$; B$ - нет; файл будет дополнен нужным числом пробелов. Оператор PRINT# может быть использован в форме PRINT #1 USING..., при этом могут быть задействованы все обычные экранные форматы PRINT USING для форматирования вывода в файл.

Вообще говоря, более экономично применять оператор PRINT#, записывая каждый раз по одной переменной. Этот метод избавляет от излишних ограничителей и позволяет затем безошибочно считывать строки любого вида. Более сложные схемы ограничителей, используемые при записи нескольких переменных одним оператором PRINT# или WRITE#, могут привести к проблемам, особенно если одна переменная будет считана как две, что приведет к потере текущей позиции в файле.

После того как все данные будут записаны в файл, просто закройте его, чтобы обезопасить содержащиеся в нем данные. Напишите CLOSE, чтобы закрыть все открытые файлы, CLOSE #1, чтобы закрыть файл #1, и CLOSE #1, #3, чтобы закрыть файлы #1 и #3. Хотя в некоторых случаях Бейсик не требует закрытия файлов, но это не соответствует описанной выше ситуации. Операторы WRITE# и PRINT# выводят данные в файловый буфер, который записывается на диск только тогда, когда он заполнен информацией. Последние введенные данные записываются на диск оператором CLOSE. Отсутствие этого оператора может привести к потере данных. Например:

```
100 OPEN "A:NEWSEQ" FOR OUTPUT AS #1   'открываем файл
110 A$ = "aaaaa"                       'готовим три строки
120 B$ = "bbbbb"
130 C$ = "ccccc"
140 WRITE #1, A$, B$, C$               'запись строк
150 CLOSE                              'очистка буфера
```

Средний уровень

MS-DOS может писать последовательные файлы как методом управляющего блока файла, так и методом дескриптора файлов. Метод FCB предоставляет функцию, специально сконструированную для записи последовательных файлов. Метод же дескриптора файлов имеет только функцию записи в файл общего назначения, но ее легко использовать и для этой цели. В любом случае важен способ, с помощью которого был открыт последовательный файл. Если данные добавляются к последовательному файлу, то должна быть использована обычная функция открытия файла. Однако если файл перезаписывается заново, то требуется функция "создания" файла. Эта функция "обрезает" файл до нулевой длины, поэтому она будет равна длине записанных в него данных.

Метод FCB. Функция 15H прерывания 21H предназначена для записи в последовательный файл. Надо подготовить управляющий блок файла и область обмена с диском, как показано в [5.3.5]. Если файл должен быть перезаписан, то его надо открыть с помощью функции 16H, которая "создает" файл, "обрезая" его до нулевой длины. Если вы откроете файл с помощью функции 0FH, то остаток старого файла останется в конце файла, если длина нового файла будет меньше старого. С другой стороны, если вы хотите добавить данные к файлу, то используйте функцию открытия файла.

После того как файл открыт, вы должны установить DS:DX на начало FCB и вызвать функцию 15H, для того чтобы вывести одну запись данных. Количество данных в записи зависит от величины, которая помещена в поле длины записи, расположенное со смещением 14 в обычном FCB; по умолчанию это значение равно 128 байтам. Если размер записи меньше, чем размер сектора диска - 512 байт, то данные будут буферизоваться до тех пор, пока их не накопится достаточно, чтобы произвести реальную запись на диск; поэтому записи в последовательный файл могут успешно записываться, даже если накопитель не включен. При закрытии файла все данные, оставшиеся в буфере, сбрасываются на диск. При возврате из функции 15H, AL равен 0, если операция успешна, 1, если диск полон, и 2, если сегмент области обмена данных слишком мал.

В следующем примере на диск выводятся 5 записей длиной 256 байт. Записи могут быть набором текстовых данных. Эти данные расположены в области памяти, помеченной меткой WORKAR. В первый момент указатель на DTA устанавливается на начало этой области, а после вывода каждой записи установка DTA меняется таким образом, чтобы он указывал на 256 байт выше. Отметим, что обычно для такой рабочей области отводится специальная область памяти (см. [1.3.1]), но в данном примере для простоты используется буфер, расположенный в сегменте данных.

```
;---в сегменте данных
WORKAR DB    2000 DUP (?)   ;буфер данных
FCB    DB    1,'FILENAMEEXT',25 DUP (0)
;---DTA должен указывать на рабочую область
       LEA   DX,WORKAR      ;DS:DX указывают на DTA
       MOV   DI,DX          ;сохраняем копию
       MOV   AH,1AH         ;функция установки DTA
       INT   21H            ;устанавливаем DTA
;---открываем файл
       MOV   AH,16H         ;номер функции
       LEA   DX,FCB         ;DS:DX указывают на FCB
       INT   21H            ;открываем файл
;---устанавливаем размер записи
       LEA   BX,FCB         ;BX указывает на FCB
       MOV   AX,256         ;размер записи - 256 байт
       MOV   [BX] + 14,AX   ;записываем в поле размера записи
;---посылаем данные в файл
       MOV   CX,5           ;число записей
NEXT_R: MOV  AH,15H         ;функция записи
       LEA   DX,FCB         ;указываем на FCB
       INT   21H            ;записываем данные
       CMP   AL,2           ;проверка на ошибки
       JE    CONTINUE       ;и их обработка
       CMP   AL,1           ;диск полон?
       JE    DISK_FULL      ;на процедуру обработки полного диска
;---перенос выполнен, переустанавливаем DTA
       ADD   DI,256         ;сдвигаемся на 1 запись
       MOV   DX,DI          ;DS:DX указывают на новый DTA
       MOV   AH,1AH         ;функция установки DTA
       INT   21H            ;установка новой позиции
       LOOP  NEXT_R         ;переходим к следующей записи
```

```
;---позднее закрываем файл
        LEA   DX,FCB        ;DS:DX указывают на FCB
        MOV   AH,10H        ;функция закрытия файла
        INT   21H           ;закрываем файл
```

Метод управляющего блока файла не слишком удобен для добавления записей в конец существующего последовательного файла. В отличие от метода дескриптора файла, который позволяет указать на конец файла, здесь вы должны манипулировать полями текущей записи и текущего блока. Нужно считать последнюю, несущую информацию, запись в DTA, а потом заполнить пустое пространство в нем первой записью данных, которые вы хотите добавить. Затем перезапишите запись на ее старое место в файле, после чего вы можете добавлять сколько хотите новых записей. Файл должен быть открыт функцией 0FH.

Метод дескриптора файла. Необходима внимательность при открытии файла для последовательного вывода методом дескриптора файла. Поскольку та же самая функция используется для записи в файл прямого доступа, то при закрытии файла его длина не устанавливается равной последней позиции файлового указателя. Возьмем, например, случай, когда текстовый файл размером 2000 байт считывается с диска, а затем в процессе обработки в памяти его длина уменьшается до 1000 байт. Если файл был открыт простой командой открытия файла (функция 3DH), то после того, как новая, более короткая, версия файла будет записана на диск и файл будет закрыт, его длина останется равной 2000 байтам, из которых новый текст будет занимать первую тысячу байтов. По этой причине при открытии последовательного файла для перезаписи надо использовать функцию 3CH прерывания 21H (см. [5.3.2]). Эта функция обычно создает новый файл, но если файл уже существует, то он "обрезается" до нулевой длины. Для добавления данных в последовательный файл надо применять обычную функцию открытия файла - 3DH прерывания 21H (см. [5.3.3]).

Рассмотрим сначала случай полной перезаписи файла. Если файл был открыт с помощью функции 3CH, файловый указатель устанавливается равным нулю, поэтому отпадает необходимость в его установке. Поместите номер файла в BX, а число записываемых байтов - в CX. Затем установите DS:DX на первый байт выводимых данных и выполните функцию 40H прерывания 21H. При возврате, если флаг переноса установлен, то была ошибка и AX содержит 5, если была ошибка дискового накопителя, и 6, если неверный номер файла. В противном случае AX будет содержать число реально записанных байтов; при несовпадении, вероятнее

всего, проблема состоит в том, что диск полон. Не забудьте о процедуре восстановления при сбоях программы, поскольку в этом случае первоначальное содержимое файла будет утеряно, так как он был "обрезан" до нулевой длины. Как проверять дисковое пространство, описано в [5.1.2]. Например:

```
;---в сегменте данных
PATH    DB    'B:FILENAME.EXT',0     ;путь к файлу
BUFFER  DB    2000 DUP (?)
;---открываем файл с помощью функции "создания"
        LEA   DX,PATH          ;DS:DX указывают на путь к файлу
        MOV   CX,0             ;атрибуты файла (здесь обычные)
        MOV   AH,3CH           ;номер функции
        INT   21H              ;открываем файл
        JC    OPEN_ERROR       ;уход по ошибке
        MOV   HANDLE,AX        ;запоминаем номер файла
;---записываем в файл 1000 байт
        MOV   AH,40H           ;номер функции
        MOV   BX,HANDLE        ;номер файла в BX
        MOV   CX,1000          ;число байтов, которое надо записать
        LEA   DX,BUFFER        ;DS:DX указывают на буфер данных
        INT   21H              ;записываем данные
        JC    OUT_ERROR        ;проверка на ошибки
        CMP   CX,2000          ;обработка ошибок
        JNE   FULL_DISK        ;при их наличии
```

Для добавления записей в последовательный файл надо открыть файл с помощью функции 3DH прерывания 21H, помещая 1 в AL, если программа будет только писать данные, и 2, если программа будет и читать, и писать. Длина файла остается неизменной, хотя он будет увеличиваться по мере добавления данных. Файловый указатель должен быть установлен на конец файла, иначе существующие данные будут перезаписаны. Это выполняется функцией 42H прерывания 21H. Поместите номер подфункции 2 в AL для установки указателя на конец файла, а номер файла поместите в BX. CX:DX указывают на смещение относительно конца файла, начиная с которого будет производиться запись, поэтому обнулите эти регистры. Затем выполните функцию установки указателя. При возврате установленный флаг переноса индицирует ошибку, при этом в AX будет находиться 1, если номер подфункции в AL был неверен, и 6, если неверно был

указан номер файла. После того как файловый указатель установлен, операция записи выполняется, как и в предыдущем случае:

```
;---в сегменте данных
PATH    DB   'B:FILENAME.EXT',0      ;путь к файлу
BUFFERDB   1000 DUP(?)
;---открываем файл
        LEA  DX,PATH            ;DS:DX указывают на путь
        MOV  AL,1               ;код открытия только для записи
        MOV  AH,3DH             ;номер функции
        INT  21H                ;открываем файл
        JC   OPEN_ERROR         ;уход по ошибке
        MOV  HANDLE,AX          ;сохраняем номер файла
;---установка файлового указателя на конец файла
        MOV  BX,AX              ;номер файла в BX
        MOV  CX,0               ;CX:DX дают смещение относительно конца
        MOV  DX,0
        MOV  AL,2               ;код для конца файла
        MOV  AH,42H             ;функция установки указателя
        INT  21H                ;устанавливаем указатель
        JC   POINT_ERROR        ;проверка на ошибку
;---добавляем к файлу 300 байт
        MOV  AH,40H             ;номер функции
        MOV  BX,HANDLE          ;номер файла в BX
        MOV  CX,300             ;число записываемых байтов
        LEA  DX,BUFFER          ;DS:DX указывают на буфер данных
        INT  21H                ;добавляем данные
        JC   OUTPUT_ERROR       ;проверка на ошибки
        CMP  CX,300             ;обработка ошибок
        JNE  FULL_DISK          ;при их наличии
```

5.4.4. Чтение из последовательных файлов

Чтение из последовательного файла мало чем отличается от записи в него, за исключением того, что этот процесс - обратный. В Бейсике данные берутся из файла и присваиваются отдельным переменным или элементам массива данных. В языке ассемблера данные помещаются в буфер, расположенный в памяти. В последнем случае данные передаются по записям и программа должна сама выделять элементы данных, составляющие записи (под

записью здесь понимается порция данных, которая считывается из файла).

Высокий уровень

Чтение последовательных файлов в Бейсике проще, чем их запись, поскольку имеется только две возможности для обращения к ним, в зависимости от того, какие символы в файле используются в качестве ограничителей элементов данных. Оператор INPUT# распознает запятые и кавычки как разделители данных, так же как и пары возврат каретки/перевод строки. Оператор LINE INPUT# распознает только комбинации CR/LF, поэтому он может использоваться для чтения целых строк текста, содержащих другие ограничители. Эта возможность удобна при обработке текстов.

Для чтения трех элементов оператором INPUT# сначала откройте файл, как обсуждалось в [5.3.3] (например, OPEN "A:NEWSEQ" FOR INPUT AS #1). Если файл был открыт под номером 1, то оператор INPUT #1, X$, Y$, Z$ присвоит значение первых трех элементов данных трем строковым переменным. При вводе числовых переменных, например INPUT #1, X, Y, Z, необходимо, чтобы соответствующие данные в файле были числовыми. Число с двойной точностью должно считываться в переменную двойной точности, с тем чтобы она могла хранить восемь байтов такого числа. Другой способ прочитать три элемента данных состоит в размещении их в массиве:

```
100 DIM ITEM$(40)                'создаем массив строк из 40 элементов
110 FOR N = 0 TO 39              'для каждого элемента
120 INPUT #1, ITEM$(N)           'считываем его и помещаем в массив
130 NEXT
```

Чтобы прочитать n-й элемент последовательного файла, программа должна прочитать все предшествующие ему элементы. Для этого надо создать цикл, в котором будут считываться элементы данных без их дальнейшего сохранения.

Оператор LINE INPUT# действует в основном аналогично оператору INPUT#, за исключением того, что он может принимать только одну и только строковую переменную. Переменная может быть длиной до 254 символов, это максимально допустимый размер строковых переменных в Бейсике. Пара возврат каретки/перевод строки, содержащаяся в файле, включается в строку, возвращаемую оператором LINE INPUT#. Это свойство позволяет обнаруживать конец параграфа в текстовом файле.

Функция EOF (конец файла) может быть использована для определения того, все ли элементы файла были прочитаны. EOF возвращает -1, если файл исчерпан, и 0 - в противном случае. В функции требуется указать номер файла, под которым он был открыт; например, если файл был открыт как #2, то X = EOF(2). В следующем примере весь текстовый файл считывается в массив.

```
100 OPEN "TEXT.AAA" FOR INPUT AS #2   'открываем файл
110 DIM TEXT$(500)                    'не более 500 строк
120 LINECOUNT = 0                     'счетчик строк
130 LINE INPUT #2, TEXT$(LINECOUNT)   'получаем строку
140 IF EOF(2) THEN 170                'проверка на конец файла
150 LINECOUNT = LINECOUNT + 1         'увеличиваем счетчик
160 GOTO 130                          'на следующую строку
170 ...                               'файл прочитан
```

Оператор INPUT$ читает из последовательного файла указанное число символов. На самой программе лежит забота о выделении отдельных элементов данных. Формат этого оператора для чтения 30 символов из файла #1 таков: S$ = INPUT$(30,#1). Хотя вы можете указывать число байтов для чтения, необходимо, чтобы это число не превосходило 254, поскольку это максимальный размер строковой переменной, в которую помещаются данные. Оператор INPUT$ полезен при передаче массы данных в непрерывную область памяти. В следующем примере первые 200 байт последовательного файла выгружаются в буфер монохромного дисплея, с тем чтобы они были выведены на экран (включая управляющие коды).

```
100 OPEN "A:NEWFILE" FOR INPUT AS #1   'открываем файл
110 CLS: DEF SEG = &HB000              'указываем на буфер
120 FOR N = 0 TO 9                     'получаем 10 групп
130 S$ = INPUT$(20,#1)                 'по 20 байт
140 FOR M = 1 TO 20                    'берем каждый байт
150 POKE N*160 + M*2, ASC(MID$(S$,M,1) 'и помещаем его в буфер
160 NEXT M                             'переход к следующему байту
170 NEXT N                             'переход к следующей группе
```

Средний уровень

Как и во всех случаях с файловыми операциями, MS-DOS может читать последовательные файлы как методом управляющего

блока файла, так и методом дескриптора файлов. Только первый из них имеет функцию, специально предназначенную для чтения последовательных файлов. Метод же дескриптора файлов использует более общую функцию, манипулируя ею особым образом, требуемым для работы с последовательными файлами.

Метод FCB. Функция 14H прерывания 21H читает последовательные файлы. Надо создать управляющий блок файла и область обмена с диском, как объяснено в [5.3.5]. Файл должен быть открыт функцией 0FH прерывания 21H (см. [5.3.3]). DS:DX должны указывать на первый байт FCB, после чего функция 14H будет читать при каждом вызове по одной записи из файла. Вы можете установить размер записи по смещению 14 в FCB. Это надо делать после того, как файл открыт, так как при открытии файла DOS по умолчанию вставляет в это поле значение, равное 128.

Каждый раз при вызове функции данные загружаются в память, начиная с первого байта DTA. Если DTA используется как небольшой временный буфер, то перед чтением следующей записи содержимое DTA должно быть перенесено в область данных файла, отведенную в памяти. Можно, наоборот, установить указатель DTA на стартовый адрес памяти, начиная с которого будет размещаться файл, а после чтения каждой записи указатель увеличивать на размер записи, с тем чтобы он отмечал то место, где должна быть следующая запись.

С помощью установки полей текущей записи (DB, смещение 1FH) и блока текущей записи (DW, смещение 0CH), отличными от нуля, последовательный файл может читаться, начиная с любого требуемого места (установка должна быть сделана после открытия FCB). После чтения каждой текущей записи ее поле автоматически увеличивается на 1, а после чтения 128 записей - увеличивается поле текущего блока. При возврате AL равен 0, что означает - вся запись успешно прочитана. При обнаружении конца файла AL будет содержать 1, если функция 14H вообще не возвратила данных, и 3, если запись прочитана частично.

В приведенном примере из файла считываются две записи и последовательно помещаются в нужную область памяти. Размер записи установлен равным 256 байтам. Записи считываются в цикле, и после прочтения первой из них указатель на DTA изменяется таким образом, чтобы он был установлен на следующий пустой байт в области данных.

```
;---помещаем FCB в сегмент данных
FCB     DB      0,'OLDDATA DAT', 25 DUP(0)
DAT_ARDB        512 DUP (?)         ;используем как DTA
;---устанавливаем DTA на начало области данных
```

```
        LEA   DX,DAT_AR        ;DS:DX указывают на DTA
        MOV   DI,DX            ;сохраняем копию
        MOV   AH,1AH           ;функция установки DTA
        INT   21H              ;устанавливаем DTA
;---открываем файл
        LEA   DX,FCB           ;DS:DX указывают на FCB
        MOV   AH,0FH           ;функция открытия файла
        INT   21H              ;открываем файл
        CMP   AL,0             ;проверка на ошибку
        JNE   OPEN_ERROR
;---устанавливаем размер записи - 256 байт
        LEA   BX,FCB           ;DS:DX указывают на FCB
        MOV   AX,256           ;размер записи
        MOV   DS:[BX] +14,AX   ;посылаем в поле размера записи
;---чтение данных
        MOV   CX,2             ;число читаемых записей
NEXTR:  MOV   AH,14H           ;функция чтения файла
        LEA   DX,FCB           ;DS:DX указывают на FCB
        INT   21H              ;читаем одну запись
        CMP   AL,0             ;все в порядке?
        JE    CONT
        CMP   AL,2             ;проверка на ошибку
        JE    READ_ERROR

.
.
CONT:   ADD   DI,256           ;увеличиваем указатель
        MOV   DX,DI            ;DX указывает на новую DTA
        MOV   AH,1AH           ;функция установки DTA
        INT   21H              ;устанавливаем DTA
        LOOP  NEXTR            ;переход к чтению следующей записи
;--- закрываем файл
        LEA   DX,FCB           ;DS:DX указывают на FCB
        MOV   AH,10H           ;функция закрытия файла
        INT   21H              ;закрываем файл
        CMP   AL,0FFH          ;проверка на ошибку
        JE    CLOSE_ERROR
```

Метод дескриптора файлов. Функция 3FH прерывания 21H может читать данные из файла последовательно. Эта функция используется для любого чтения из файла с помощью метода деск-

риптора файлов, включая файлы прямого доступа. Файл должен быть открыт функцией 3DH прерывания 21H с кодом 0 в AL, если он открывается только для чтения, и с кодом 2, если он открывается для чтения и записи. При открытии файловый указатель автоматически устанавливается на первый байт файла. Функция чтения из файла указывает, сколько байтов должно быть считано, и после того, как это сделано, файловый указатель устанавливается на байт, следующий за последним считанным байтом, подготавливая таким образом очередное обращение к функции. Отметим, что файловый указатель уникален для каждого файла: операции над другими файлами не меняют его позицию.

Программа может создать небольшой временный буфер размером, скажем, 512 байт, и постоянно вызывать функцию чтения, не заботясь о позиции файлового указателя. Другой метод состоит в считывании всего файла прямо в то место памяти, где он должен быть расположен. В этом случае надо просто потребовать, чтобы функция прочитала больше байтов, чем реально содержится в файле, так как чтение прекращается при достижении последнего байта файла. Однако вам необходимо знать точную длину файла, чтобы знать, где кончаются данные в буфере, в который вы считали файл.

Размер файла можно определить, сдвинув файловый указатель на конец файла. Это надо сделать сразу же после открытия файла. Поместите в AL код 2 и вызовите функцию 42H, чтобы сдвинуть указатель на конец файла. CX и DX должны содержать 0, так как в противном случае указатель будет сдвинут с конца файла на величину, которая содержится в этих регистрах. При возврате DX:AX будут содержать новую позицию указателя как смещение относительно начала файла, т.е. в данном случае - длину файла. Не забудьте снова вернуть файловый указатель на начало файла, перед тем как читать его; это делается точно так же, с той лишь разницей, что в AL надо поместить 0. Если при выполнении функции 42H возникает ошибка, то устанавливается флаг переноса, а в AX возвращается 1, если неверен номер функции, и 6, если указан неверный номер файла.

Теперь программа готова для чтения файла. Надо поместить номер файла в BX, а требуемое число байтов в CX и выполнить прерывание. При возврате AX будет содержать число реально прочитанных байтов. Если AX равен нулю, то достигнут конец файла. При других ошибках устанавливается флаг переноса, а AX содержит 5 при ошибке оборудования и 6, если указан неверный номер файла. В следующем примере в буфер памяти считывается весь небольшой файл. Для удобства буфер располагается в сегменте данных, что существенно увеличивает размер программы на

диске. В своих программах лучше создавать буфер, используя технику распределения памяти, описанную в [1.3.1].

```
;---в сегменте данных
PATH    DB    'A:FILENAME.EXT'0   ;строка пути к файлу
BUFFERDB      1000 DUP (?)        ;буфер данных
HAN     DW    ?                   ;номер файла
SIZE    DW    ?                   ;размер файла
;---открываем файл
        LEA   DX,PATH             ;DS:DX указывают на путь
        MOV   AL,0                ;код открытия для чтения
        MOV   AH,3DH              ;функция открытия файла
        INT   21H                 ;открываем файл
        JC    OPEN_ERROR          ;проверка на ошибку
        MOV   HAN,AX              ;запоминаем номер файла
;---устанавливаем файловый указатель на конец файла
        MOV   AH,42H              ;функция установки указателя
        MOV   AL,2                ;код конца файла
        MOV   BX,HAN              ;номер файла
        MOV   CX,0                ;смещение равно нулю
        MOV   DX,0
        INT   21H                 ;устанавливаем указатель
        JC    POINT_ERROR1        ;обработка ошибки
        MOV   SIZE,AX             ;запоминаем размер (меньше 64K)
;---возвращаем указатель на начало
        MOV   AH,42H              ;номер функции
        MOV   AL,0                ;код начала файла
        MOV   CX,0                ;смещение равно нулю
        MOV   DX,0
        INT   21H                 ;устанавливаем указатель
        JC    POINT_ERROR2        ;обработка ошибки
;---читаем весь файл
        MOV   AH,3FH              ;номер функции чтения файла
        MOV   BX,HAN              ;номер файла
        MOV   CX,SIZE             ;число считываемых байтов
        LEA   DX,BUFFER           ;DS:DX указывают на буфер
        INT   21H                 ;читаем файл
        JC    READ_ERROR          ;обработка ошибки
;---закрываем файл
        MOV   BX,HAN              ;номер файла
        MOV   AH,3EH              ;функция закрытия файла
        INT   21H                 ;закрываем файл
        JC    CLOSE_ERROR         ;обработка ошибки
```

5.4.5. Запись в файлы прямого доступа

Физически файлы прямого доступа ничем не отличаются от последовательных файлов, за исключением режима доступа. Файл прямого доступа предполагает, что его данные организованы в виде записей фиксированной длины, таким образом, положение каждой записи может быть вычислено (в последовательных файлах n-й элемент ищется путем подсчета разделителей между элементами, начиная с начала файла). Операционная система автоматически выполняет эти вычисления. Однако любая программа может выполнять эту работу сама, устанавливая файловый указатель на нужную позицию и считывая последовательно такое число байтов, которое образует запись.

Высокий уровень

В [5.3.3] объяснен формат открытия файлов прямого доступа в Бейсике. В отличие от последовательного файла файл прямого доступа может читаться и записываться в одно и то же время, без закрытия и повторного его открытия. Оператор OPEN завершается числом, представляющим размер записи файла. Например, OPEN "R", 1, "NEWDATA", 20 устанавливает для файла NEWDATA размер записи в 20 байт (при этом файл открывается как файл #1).

После того как файл открыт, его записи могут быть разбиты на составляющие переменные с помощью оператора FIELD. Оператор FIELD указывает, сколько байтов записи отводится под каждую переменную. Например, запись длиной 20 байт может быть разбита оператором FIELD 1, 14 AS LASTNAME$, 2 AS DEPOSIT$, 4 AS ACCTNUM$. В этом операторе первая цифра 1 указывает, что данный оператор FIELD описывает разбиение записи для файла, открытого под номером #1. Данные в записи располагаются в том порядке, в каком они описаны в операторе FIELD. Операторы RSET и LSET сдвигают данные в полях, выравнивая их по правому (RSET) или левому (LSET) краю и заполняя остающиеся пустые места пробелами. Например, для того чтобы вставить фамилию "SMITH" в 14-байтовое поле с именем LASTNAME$, надо записать RSET LASTNAME$ = "SMITH", или если переменной N$ было присвоено значение "SMITH", то RSET LASTNAME$ = N$. Вместо RSET может быть использовано LSET. Когда впоследствии данные считываются из поля в переменную, то ей присваиваются все 14 байт. При использовании RSET программа удалит все лишние пробелы в начале строковой пере-

менной, однако, если будет использоваться LSET, пробелы будут удаляться справа.

Отметим, что все имена переменных в операторе FIELD относятся к строковым переменным. В файлах прямого доступа Бейсик рассматривает все переменные, включая числовые, как строковые. Числовая переменная должна быть преобразована к специальному виду, прежде чем ее значение может быть присвоено полю, а когда она затем считывается из поля, необходимо обратное преобразование. Слово *преобразование* стоило бы заключить в кавычки, поскольку Бейсик на самом деле не меняет способ представления числа в памяти; он просто обрабатывает число особым образом. Числовые поля требуют два байта для целых чисел, четыре байта для чисел с обычной точностью и восемь байтов для чисел с двойной точностью. Такое же число байт требуется и для представления этих чисел в памяти. Для преобразования их в строковую форму надо использовать функции MKI$, MKS$ и MKD$, которые осуществляют преобразование число-строка для целых, вещественных и чисел с двойной точностью соответственно. Обычно эти функции комбинируются с операторами RSET или LSET, например RSET = ACCTNUM$ = MKI$(X), где X - целая переменная, если полю ACCTNUM$ было отведено два байта в операторе FIELD.

После того как поля заполнены операторами RSET и LSET, запись выводится на диск с помощью оператора PUT#. PUT #1, 245 помещает данные в запись номер 245 файла, открытого под номером #1. Номер записи может быть опущен, в этом случае данные помещаются в запись с номером на единицу больше, чем номер последней записи (начиная с записи 1). Запись выводится вся целиком, даже если не все поля были заполнены данными. Отметим, что поля буфера не очищаются при выполнении операции PUT, поэтому элементы данных, например текущая дата, могут помещаться в буфер только один раз, а затем они будут помещены во все записи, которые будут выводиться в течение данного сеанса работы. Функция LOC возвращает номер последней выведенной в файл записи. Если файл был открыт под номером #3, напишите X = LOC(3).

Функция LOF (длина файла) возвращает длину файла в байтах. Для определения числа записей, содержащихся в файле, надо разделить это значение на длину записи. Добавление 1 к этому значению дает номер записи, который надо использовать, чтобы добавить к файлу новые записи. Если файл был открыт под номером #2, а длина его записей равна 32 байтам, то требуемое значение вычисляется как RECORDNUM = LOF(2)/32 + 1.

В следующем примере файл прямого доступа открывается с длиной записи, равной 24 байтам, причем запись разбита на три

переменные. Программа запрашивает пользователя о содержимом всех трех полей, а когда все они введены, добавляет запись к файлу. В строке 120 вычисляется начальный номер записи. Отметим, что каждый раз при выполнении оператора PUT данные могут не записываться физически на диск. В выходном буфере могут накапливаться несколько записей и только затем выводиться на диск.

```
100 OPEN "R", 1, "A:NEWDATA.DAT", 24  'открываем файл
110 FIELD 1, 18 AS LASTNAME$, 2 AS AGE$, 4 AS WEIGHT
120 R = LOF(1)/24 + 1                 'номер последней записи + 1
130 CLS                               'чистим экран
140 INPUT "Enter name:",N$            'получаем имя (строка)
150 INPUT "Enter age:",A%             'получаем возраст (целое)
160 INPUT "Enter weight:",W!          'получаем вес (вещественное)
170 RSET LASTNAME$ = N$               'помещаем в поле имя
180 RSET AGE$ = MKI$(A%)              'помещаем в поле возраст
190 RSET WEIGHT$ = MKS$(W!)           'помещаем в поле вес
200 PUT #1, R                         'выводим запись
210 R = R + 1                         'увеличиваем счетчик
220 PRINT: PRINT "Do another (y/n)?"  'запрос пользователя
230 C$ = INKEY$: IF C$ = "" THEN 220  'ожидаем ответа
240 IF C$ = "y" THEN CLS: GOTO 130    'если да, то на начало
250 CLOSE                             'иначе - закрываем файл
```

Средний уровень

Как и при всех других операциях с файлами, в MS-DOS имеются два метода записи в файл прямого доступа: один - с использованием управляющего блока файла, а другой - с помощью дескриптора файла. В обоих случаях вы должны создать буфер обмена данными, размер которого должен быть не меньше чем размер записи.

Метод управляющего блока файла. Откройте управляющий блок файла с помощью функции 0FH и пусть DS:DX указывают на него. После того как файл открыт, поместите номер записи для прямого доступа в поле записи прямого доступа FCB. Затем вызовите функцию 22H прерывания 21H, которая передаст данные из DTA в файловый буфер, организованный при создании FCB. Данные могут быть записаны на диск не сразу, если размер записи меньше размера буфера. Реальная запись на диск будет проис-

ходить тогда, когда очередной вызов функции 22H заполнит буфер.

При возврате из функции 22H AL будет содержать 00, если обмен прошел успешно. В противном случае в нем будет находиться 1, если не хватает пространства на диске, и 2, если область переноса мала для того, чтобы вывести одну запись (т.е. если размер буфера, установленный системой, меньше, чем тот, который указан в FCB).

```
;---в сегменте данных
FCB DB    1, 'NEWDATA ', 25 DUP (0)
DTA DB    256 DUP (?)
;---открываем файл и устанавливаем поля FCB
    MOV   AH,0FH        ;номер функции
    LEA   DX,FCB        ;DS:DX указывают на FCB
    MOV   BX,DX         ;копируем смещение для FCB
    INT   21H           ;открываем файл
    MOV   AX,256        ;размер записи
    MOV   [BX] + 14,AX  ;помещаем в поле размера записи
    MOV   AX,233        ;номер записи
    MOV   [BX] + 33,AX  ;помещаем в поле номера записи
    MOV   AX,0          ;обнуляем старший байт этого слова
    MOV   [BX] + 35,AX
;---перенос данных из DTA в файл
    MOV   AH,22H        ;номер функции записи с прямым доступом
    LEA   DX,FCB        ;DS:DX указывают на FCB
    INT   21H           ;записываем данные
    CMP   AL,0          ;проверка на ошибку
    JNE   WRITE_ERROR
;---закрываем файл
    LEA   DX,FCB        ;DS:DX указывают на FCB
    MOV   AH,10H        ;функция закрытия файла
    INT   21H           ;закрываем файл
    CMP   AL,0FFH       ;проверка на ошибку
    JE    CLOSE_ERROR
```

Часто программа работает сразу с несколькими записями прямого доступа, передавая их в память и из памяти как единое целое. При использовании метода FCB MS-DOS предоставляет для этого специальную функцию, называемую *запись блока с прямым доступом*. Это функция 28H прерывания 21H. При входе DS:DX должны указывать на открытый FCB, в котором поле записи прямого доступа должно быть равно номеру первой из выводимых записей. Эта функция совершенно аналогична функции в приведенном выше примере. Единственное отличие (кроме номера

функции) состоит в том, что в CX должно быть указано число записей в блоке (не путайте эти "блоки" с блоками по 128 записей, с помощью которых система находит требуемую запись - программа может читать любое число записей, начиная с любого места).

В CX возвращается число реально прочитанных записей. AL будет содержать 0, если все записи успешно выведены, 1, если не хватает пространства на диске (при этом не будет выведена ни одна запись). В отличие от 22H эта функция автоматически увеличивает поля текущей записи, текущего блока и записи прямого доступа в FCB, так что они будут указывать на запись, следующую за последней прочитанной. Отметим, что если при выполнении этой функции сделать CX = 0, размер файла будет установлен в соответствии с числом записей, равным полю записи прямого доступа. Таким образом можно резервировать дисковое пространство для файла.

Метод дескриптора файлов. При использовании для доступа метода дескриптора файлов система не различает последовательные файлы и файлы прямого доступа. Ваша программа должна вычислить позицию в файле, с которой начинается требуемая запись, и установить на нее файловый указатель. Файловый указатель позиционируется с помощью функции 42H прерывания 21H. Поместите номер файла в BX, а смещение в файле - в CX:DX (CX будет содержать старший байт значения). Затем поместите в AL кодовый номер от 0 до 2. При AL = 0 указатель будет установлен со смещением CX:DX байтов относительно начала файла; при AL = 1 указатель будет установлен со смещением CX:DX относительно текущей позиции, а при AL = 2 указатель будет установлен со смещением CX:DX относительно конца файла (т.е. таким образом файл будет расширен). Отрицательные числа недопустимы в качестве смещений. При возврате DX:AX будут содержать новое положение указателя (старший байт в DX). Если устанавливается флаг переноса, то произошла ошибка. В этом случае AX будет содержать 1, если указан неверный код в AL, и 6, если указан неверный номер файла.

После позиционирования файлового указателя запись прямого доступа выводится с помощью той же функции 40H прерывания 21H, которая использовалась для вывода в последовательный файл. При входе BX содержит номер файла, а CX - число байтов, которое надо записать. При возврате AX будет содержать число реально записанных байтов. Если оно отличается от числа, помещенного в CX, то, вероятно, диск полон (см. [5.1.4]). Как обычно, при возникновении ошибки устанавливается флаг переноса. В этом случае AX будет содержать 5 при ошибке накопителя и 6, если указан неверный номер файла.

Файловый указатель играет ту же роль для образа файла на диске, что и DTA для образа файла в памяти. Он может сдвигаться как угодно для доступа к различным частям файла. Будьте внимательны, манипулируя файловым указателем при работе с файлом прямого доступа - содержимое любого поля любой записи может быть легко извлечено с диска и помещено в требуемое место в памяти.

```
;---в сегменте данных
HAN    DW  ?                ;номер файла
PATH   DB  'A:NEWDATA',0    ;строка пути к файлу
BUFER  DB  30 DUP (?)       ;буфер выводимых записей
;---открываем файл
       MOV AH,3DH           ;номер функции
       MOV AL,1             ;код открытия для записи
       LEA DX,PATH          ;DS:DX указывают на путь
       INT 21H              ;открываем файл
       JC  OPEN_ERROR       ;проверка на ошибку
       MOV HAN,AX           ;сохраняем номер файла
;---вычисляем позицию записи и устанавливаем файловый указатель
       MOV AX,30            ;размер записи 30 байт
       MOV CX,54            ;номер записи #54 (55-я запись)
       MUL CX               ;теперь смещение для нее в DX:AX
       MOV CX,DX            ;помещаем старшее слово в DX
       MOV DX,AX            ;помещаем младшее слово в CX
       MOV AL,0             ;устанавливаем указатель на начало
       MOV AH,42H           ;функция установки указателя
       MOV BX,HAN           ;номер файла
       INT 21H              ;устанавливаем указатель
       JC  POINTER_ERROR    ;проверка на ошибку
;---выводим запись с прямым доступом
       MOV AH,40H           ;номер функции
       MOV BX,HAN           ;номер файла
       MOV CX,30            ;размер записи
       LEA DX,BUFER         ;DS:DX указывают на буфер
       INT 21H              ;выводим запись
       JC  WRITE_ERROR      ;проверка на ошибку
```

В отличие от метода FCB метод дескриптора файлов не предоставляет специальной функции для вывода блока записей прямого

доступа. Однако вашей программе необходимо только вычислить количество составляющих блок записей байтов, которое должно быть выведено.

5.4.6. Чтение из файлов прямого доступа

Чтение из файлов прямого доступа является обратным процессом по отношению к их выводу. MS-DOS вычисляет позицию в файле на диске, затем считывает запись и помещает ее в память. Далее программа должна разделить запись на поля того же размера, который был использован при конструировании записи. Не забудьте удалить символы пробела, добавленные при заполнении полей. Обсуждение процесса вывода данных в файлы прямого доступа (см. [5.4.5]) поможет вам лучше понять содержание данного параграфа.

Высокий уровень

Для чтения файла прямого доступа необходимо открыть его и определить поля записи, как объяснено в параграфе, относящемся к записи в файлы прямого доступа. Затем надо использовать оператор GET # для чтения определенной записи с диска. GET #1,23 считывает запись номер #23 из файла, открытого под номером #1. При чтении записи переменной, именованной в операторе FIELD, автоматически присваивается соответствующее значение из записи. Например, если оператор FIELD имеет вид FIELD 1, 20 AS X$, 2 AS Y$, то после выполнения оператора GET 1,23 переменной X$ будет присвоено значение первых 20 байт записи 23, а переменной Y$ - следующих 10 байт. Операторы, аналогичные RSET и LSET, применяемые для выделения полей данных, отсутствуют.

В случае числовых полей напоминаем, что они должны быть преобразованы в строковый вид с помощью функций MKI$, MKS$ и MKD$. Для восстановления их оригинальных значений, с тем чтобы над ними можно было проводить операции и печатать их, надо преобразовать эти строки с помощью функций CVI, CVS и CVD. Если Y$ содержит целое число, то для выполнения обратного преобразования запишите Y% = CVI(Y$), при этом переменная Y% будет содержать значение, которое она имела перед тем, как была специально обработана для вывода в файл прямого доступа. Если вы выведете строковое значение переменной, то увидите, что это число в интервале от 0 до 65535, закодированное в два символа ASCII.

В следующем примере открывается файл, созданный в примере из [5.4.5], и выводится содержимое любой из затребованных записей.

```
100 OPEN "A:NEWDATA" AS #1 LEN = 24    'открываем файл
110 FIELD 1, 18 AS LASTNAME$, 2 AS AGE$, 4 AS WEIGHT$
120 CLS: INPUT "What is the record number";R  'запрос записи
130 IF R*24 > LOF(1) THEN BEEP: PRINT"No such record": GOTO 120
140 GET #1,R                           'читаем запись из файла
150 PRINT LASTNAME$, CVI(AGE$), CVS(WEIGHT$)  'выводим ее
160 PRINT: PRINT "Do another (y/n)?"   'будем повторять?
170 C$ = INKEY$: IF C$ = "" THEN 170   'ожидаем ввода
180 IF C$ = "y" OR C$ = "Y" THEN 120   'повторяем при необходимости
190 CLOSE                              'иначе - закрываем файл
```

Средний уровень

Метод FCB доступа к файлам имеет две функции для чтения записей с прямым доступом. Метод же дескриптора файлов использует ту же функцию, что и для чтения последовательных файлов. Каждый из методов доступа будет рассматриваться отдельно.

Метод FCB. Функция 21H прерывания 21H читает по одной записи из файла прямого доступа, в то время как вторая функция, 27H, может читать блок последовательных записей. Создайте управляющий блок файла, как показано в [5.3.5], и откройте его (см. [5.3.3]). После того как FCB открыт, введите в него значения полей размера записи (DW по смещению 14) и номера записи прямого доступа (DD по смещению 33). Если DS:DX указывают на первый байт FCB, то можно вызывать функцию 21H для чтения записи, которая будет помещена в память, начиная с первого байта DTA.

Если запись успешно прочитана, то в AL будет возвращен 0. Однако при этом нет гарантии, что чтение прошло без ошибок, поскольку неверный размер записи может привести к тому, что части прилегающих записей будут считаны, как будто это одна запись. Если запрошена запись с номером большим, чем число записей в файле, то в AL будет возвращено 1 или 3. Возвращение кода 3 означает, что был считан самый конец файла и была прочитана часть записи данных, возвращение кода 1 - что данные вообще не были считаны.

В следующем примере считывается одна запись и помещается в DTA:

```
;---в сегменте данных
FCB  DB    1,'OLDDATA  ', 25 DUP (0)
;---открываем файл и устанавливаем поля FCB
     MOV   AH,0FH        ;номер функции
     LEA   DX,FCB        ;DS:DX указывают на FCB
     MOV   BX,DX         ;копируем смещение FCB
     INT   21H           ;открываем файл
     MOV   AX,55         ;размер записи 55 байт
     MOV   [BX] + 14,AX  ;помещаем в поле размера записи
     MOV   AX,22         ;номер записи для чтения
     MOV   [BX] + 33,AX  ;помещаем в поле номера записи
     MOV   AX,0          ;обнуляем старшее слово этого поля
     MOV   [BX] + 35,AX
;---перенос данных из файла в DTA
     MOV   AH,21H        ;номер функции чтения с прямым доступом
     LEA   DX,FCB        ;DS:DX указывают на FCB
     INT   21H           ;читаем данные, помещая их в DTA
     CMP   AL,0          ;проверка на ошибку
     JNE   READ_ERROR
;---закрываем файл
     MOV   AH,10         ;номер функции закрытия файла
     LEA   DX,FCB        ;DS:DX указывают на FCB
     INT   21H           ;закрываем файл
```

Для чтения блока последовательных записей в память за один прием применяют функцию 27H прерывания 21H. Ее выполнение подготавливается точно так же, как и функции 21H, за исключением того, что CX должен еще содержать число записей, которые надо прочитать за один прием. При возврате CX будет содержать число реально прочитанных записей. Значения возвращаемые в AL совпадают с теми, которые возвращаются функцией 21H. В отличие от функции 21H поля FCB, в которых хранится информация о положении записи (поле записи прямого доступа, текущего блока и текущей записи), автоматически увеличиваются, с тем чтобы они после выполнения функции указывали на следующую несчитанную запись.

Отметим, что как в случае чтения одной, так и в случае чтения нескольких записей поля текущего блока и текущей записи FCB устанавливаются по значению поля записи прямого доступа. Если вам известны значения текущего блока и текущей записи, а не соответствующий номер записи прямого доступа, то исполь-

зуйте функцию 24H прерывания 21H, чтобы она проделала за вас все вычисления. У этой функции нет входных регистров, надо только, чтобы DS:DX указывали на открытый FCB. При возврате поле записи прямого доступа будет заполнено значением, соответствующим установке двух других полей.

Метод дескриптора файлов. Выше было показано, как выводить записи прямого доступа с помощью метода дескриптора файлов. Процедура чтения из файла с прямым доступом подготавливается аналогичным образом: путем вычисления смещения в файле, на которое должен быть установлен файловый указатель. DS:DX должны указывать на буфер, в который будет помещена запись, после чего надо выполнить функцию 3FH прерывания 21H. При входе CX должен содержать размер записи, а BX - номер файла.

```
;---в сегменте данных
HAN    DB   ?
PATH   DB    A:OLDDATA ,0
BUFER DB   30 DUP(?)
;---открываем файл
       MOV AH,3DH      ;номер функции
       MOV AL,0        ;код открытия для чтения
       LEA  DX,PATH    ;DS:DX указывают на путь к файлу
       INT  21H        ;открываем файл
       JC   OPEN_ERR   ;проверка на ошибку
       MOV HAN,AX      ;запоминаем номер файла
;---вычисляем позицию записи и устанавливаем файловый указатель
       MOV AX,30       ;размер записи
       MOV CX,54       ;читаем запись #54 (55-ю запись)
       MUL CX          ;смещение записи в DX:AX
       MOV CX,DX       ;помещаем старшее слово смещения в DX
       MOV DX,AX       ;помещаем младшее слово смещения в CX
       MOV AL,0        ;устанавливаем указатель на начало файла
       MOV AH,42H      ;функция установки указателя
       MOV BX,HAN      ;номер файла
       INT  21H        ;устанавливаем указатель
       JC   POINT_ERR  ;обработка ошибки
;---читаем запись с прямым доступом
       MOV AH,3FH      ;номер функции
       MOV BX,HAN      ;номер файла
       MOV CX,30       ;размер записи
       LEA  DX,BUFER   ;DS:DX указывают на буфер для записи
```

```
        INT  21H          ;читаем запись
        JC   READ_ERR     ;обработка ошибки
;---закрываем файл
        MOV BX,HAN        ;номер файла
        MOV AH,3EH        ;функция закрытия файла
        INT  21H          ;закрываем файл
        JC   CLOSE_ERR    ;проверка на ошибку
```

5.4.7. Проверка данных после операций чтения/записи

MS-DOS может проверять правильность производимого обмена с диском непосредственно во время обмена. Ошибки происходят настолько редко, что средства проверки обычно не используются, чтобы не замедлять обмен с диском. Однако в случае необходимости имеются два способа проверки. Один состоит во включении команды VERIFY = ON в файл CONFIG.SYS, который автоматически читается при загрузке операционной системы. Впоследствии все дисковые операции будут проверяться. Это единственный способ проверки, доступный в Бейсике. Второй метод состоит в использовании специальной функции DOS для верификации только критических дисковых операций. Если процедура верификации обнаруживает ошибку, она сообщает об условии критической ошибки, как описано в [7.2.5].

Средний уровень

Функция 2EH прерывания 21H включает и выключает проверку. Для включения верификации поместите в AL 1, для выключения - 0. DL также должен быть равен 0. Затем надо выполнить прерывание. У функции нет выходных регистров.

```
;---включение верификации
MOV  AL,1          ;номер кода
MOV  DL,0          ;необходимый входной регистр
MOV  AH,2EH        ;номер функции
INT  21H           ;включаем проверку
```

Для определения текущего режима верификации надо вызвать функцию 54H прерывания 21H. У нее нет входных регистров. При возврате AL = 1, если проверка включена, и AL = 0, если выключена.

5.4.8. Определение дисковых ошибок и восстановление после них

Дисковые операции очень сложны, поэтому возможно большое количество ошибок. Большинство дисковых ошибок обсуждается вместе с операциями, при которых они могут возникать. Здесь же они собраны вместе, чтобы помочь вам при разработке процедуры общего назначения для восстановления после дисковых ошибок.

Дисковые ошибки бывают двух типов: так называемые мягкие (soft) и жесткие (hard). Мягкие ошибки возникают из-за неправильного запроса на доступ к файлу (запрошенный файл может отсутствовать или дисковое пространство может кончиться прежде, чем будет записан весь файл). Жесткие ошибки возникают из-за неверного порядка или временных несоответствий при дисковых операциях, которые могут быть следствием неправильного выравнивания или проблем с накопителем. В этом случае лучше всего произвести сброс диска перед обработкой.

Высокий уровень

В [7.2.5] объяснено, как подготовить процедуру обработки ошибок. Оператор ON ERROR GOSUB заставляет программу перейти на процедуру обработки ошибки при возникновении критической ошибки. Процедура прежде всего определяет кодовый номер ошибки в Бейсике. Ниже перечислены коды дисковых ошибок:

52 *Bad file number.* (Неверный номер файла.) Файл открыт не под тем номером, к которому идет обращение (#1, #2 и т.д.).

53 *File not found.* (Файл не найден.) Используется при выполнении операторов LOAD, KILL, NAME, FILES и OPEN.

54 *Bad file mode.* (Неверный режим доступа.) Попытка доступа к файлу, открытому с другой целью, например попытка записи в последовательный файл, открытый для чтения.

55 *File already open.* (Файл уже открыт.) Попытка открыть файл, который уже открыт, или уничтожить (KILL) файл, который еще не закрыт.

58 *File already exists.* (Файл уже существует.) Попытка переименовать файл (с помощью NAME) на имя, которое уже есть в каталоге.

61 *Disk full.* (Диск полон.) См. специальное обсуждение в [5.1.4], относящееся к этой ошибке.

62 *Input past end.* (Чтение за концом файла.) Попытка прочитать из последовательного файла больше переменных, чем он содержит. Чтобы избежать этой ошибки, используйте функцию EOF, как объяснено в [5.4.4].

63 *Bad record number.* (Неверный номер записи.) Попытка прочитать или вывести запись с номером большим, чем число записей в файле.

64 *Bad file name.* (Неверное имя файла.) Используется операторами KILL, NAME и FILES.

67 *Too many files.* (Слишком много файлов.) В каталоге больше нет места для записи информации о файлах. Другой возможный вариант состоит в том, что открытие еще одного файла приведет к тому, что будет превышено максимально допустимое число одновременно открытых файлов.

70 *Disk is write-protected.* (Диск защищен от записи.)

71 *Disk is not ready.* (Диск не готов.) Наиболее вероятно, не закрыт дисковод с дискетой.

72 *Disk media error.* (Диск поврежден.) Как правило, это сообщение выдается при повреждении дискеты, однако иногда оно появляется при сбоях оборудования.

74 *Specified wrong disk in RENAME operation.* (Указан неверный диск в операции RENAME.)

75 *Path/file access error.* (Ошибка доступа к файлу.) Попытка открыть подкаталог или метку тома как файл. Или попытка писать в файл, который защищен от записи. Эта ошибка чаще всего выдается при попытке удалить текущий каталог. Появляется при операциях OPEN, NAME, MKDIR, CHDIR и RMDIR.

76 *Path not found.* (Путь не найден.) Неправильно указан путь или его не существует. Появляется при операциях OPEN, MKDIR, CHDIR и RMDIR.

После того как процедура распознала ошибку, необходимо информировать об ошибке пользователя. Когда пользователь сообщит, что причина ошибки устранена, оператор RESUME возвращает программу назад на ту строку, где произошла ошибка. Оператор RESUME может сопровождаться номером строки, поэтому программа может вернуться к началу всей последовательности дисковых операций, независимо от того, в какой строке произошла ошибка (отметим, что файлы не закрываются при возникновении ошибки). В следующем примере программа позволяет восстановить ситуацию после ошибок, связанных с переполнением диска и защитой от записи:

```
100 ON ERROR GOSUB 5000          'включение обработки ошибок
.
.
600 'здесь начинаются дисковые операции
.
.
5000 'подпрограмма обработки ошибок
5010 IF ERR = 61 PRINT "Disk full": GOTO 5100
5020 IF ERR = 70 PRINT "Disk is write protected": GOTO 5100
```

```
5100 PRINT "Correct the problem, then strike any key"
5110 C$ = INKEY$: IF C$ = "" THEN 5110
5120 RESUME 600
```

Средний уровень

Функция 1 прерывания 13H возвращает в AL байт, сообщающий статус дискового накопителя. Значения его битов могут быть следующими:

биты	0-1	01 = неверная команда, или, если бит 3 = 1, то попытка обмена данными за границей 64K
		10 = адресная метка не найдена
		11 = попытка записи на защищенный от записи диск
	2	1 = указанный сектор не найден
	3	1 = переполнение DMA (потеря данных при обмене), или, если бит 0 = 1, попытка обмена данными за границей 64K
	4	1 = данные прочитаны неверно, надо повторить
	5	1 = ошибка контроллера
	6	1 = ошибка операции поиска
	7	1 = нет ответа от накопителя (тайм-аут)

Каждая из функций обращения к диску MS-DOS возвращает только часть из возможных кодов ошибок, а некоторые функции не сообщают об ошибке. Однако во всех случаях при возникновении ошибки устанавливается флаг переноса. Если произошла ошибка, то номер кода этой ошибки возвращается в AX. Ниже перечислены коды, относящиеся к дисковым операциям:

1 Неверный номер функции
2 Файл не найден
3 Путь не найден
4 Уже открыто максимально допустимое число файлов
5 Отрицание доступа (ошибка оборудования)
6 Неверный номер файла
15 Указан неверный накопитель
16 Попытка удалить текущий каталог
17 Не то устройство
18 Больше нет файлов (при поиске в каталоге с использованием "джокеров")

Восстановление после этих "мягких" ошибок не сложно. Некоторые из них предупреждают вас о программных ошибках. Другие возникают из-за ошибочных действий пользователя. Если же не отвечает сам накопитель, то произошла критическая ошибка. В [7.2.5] показано, как написать процедуру обработки критических ошибок.

В MS-DOS 3.0 введены расширенные коды ошибок. Они могут быть получены с помощью функции 59H прерывания 21H, когда флаг переноса индицирует возникновение ошибки. Обсуждение этого вопроса см. в [7.2.5].

Глава 6. Принтер

6.1. Управление работой принтера

MS-DOS может работать с тремя параллельными устройствами (LPT1 - LPT3), и в этой главе показано, как управлять ими. Последовательные принтеры управляются так же, как и параллельные, за исключением способа, с помощью которого данные посылаются на принтер; эта информация приведена в разделе 1 главы 7. Каждое параллельное устройство имеет свой адаптер. Адаптер управляется тремя регистрами ввода/вывода; адреса портов этих регистров различны для каждого адаптера. Область данных BIOS содержит базовые адреса для каждого адаптера. Базовый адрес соответствует младшему адресу группы из трех адресов портов. Базовый адрес для LPT1 находится в ячейке 0040:0008, для LPT2 - 0040:000A и т.д. Как видно из приведенной ниже таблицы, какой адаптер назначен какому номеру LPT не определено. По этой причине программа, которая прямо адресуется в параллельный порт, должна выискивать используемые им адреса. Отметим, что при инициализации базовому адресу присваивается значение 0, когда соответствующий адаптер не установлен.

Адаптер	Регистр выходных данных	Регистр статуса	Регистр управления
Монохромная карта (PC/XT/AT)	3BCH	3BDH	3BEH
Адаптер принтера PC/XT			
Адаптер принтера PCJr	378H	379H	37AH
Последовательная/параллельная карта AT (установленная как LPT1)			

Последовательная/параллельная карта AT (установленная как LPT2)	278H	279H	27AH

Регистр выходных данных - тот адрес порта, через который проходит каждый байт данных, посылаемый в принтер. Регистр статуса сообщает различную информацию о принтере; процессор может постоянно опрашивать его, чтобы распознать момент, когда все в порядке и можно посылать данные. Регистр статуса сообщает также, что произошла ошибка на принтере. Регистр управления инициализирует адаптер и управляет выводом данных. Он может также подготавливать параллельный порт для операций прерывания, с тем чтобы принтер посылал прерывание к процессору, когда он готов к приему очередного символа, оставляя процессор свободным для других дел. Ниже перечислены значения битов регистров статуса и управления:

Регистр управления

бит 0 0 = нормальная установка, 1 = вызывает вывод байта данных
1 0 = нормальная установка, 1 = автоматический перевод строки после возврата каретки
2 0 = инициализировать порт принтера, 1 = нормальная установка
3 0 = отмена выбора принтера, 1 = нормальная установка
4 0 = прерывание принтера запрещено, 1 = разрешено
5-7 не используются

Регистр статуса

бит 0-2 не используются
3 0 = ошибка принтера, 1 = нет ошибки
4 0 = принтер off-line, 1 = принтер on-line
5 0 = бумага вставлена, 1 = нет бумаги
6 0 = принтер подтверждает прием символа, 1 = нормальная установка
7 0 = принтер занят, 1 = принтер свободен

Не может быть никаких оснований для того, чтобы в программе отсутствовала процедура восстановления после ошибок, возникающих при работе с принтером. Грамотно написанная программа должна начинать работу с проверки того, что принтер готов к работе с машиной (on-line)*. Если присоединен не один принтер, то программа должна предложить пользователю выбрать, с каким

*Английский термин on-line подразумевает, что устройство готово обмениваться информацией с машиной. В настоящей книге для обозначения этого факта мы будем пользоваться термином *связан с машиной*. - *Примеч. пер.*

из них он будет работать. Кроме того, процедура печати должна восстанавливать ситуацию при любых ошибках принтера, при этом хотелось бы, чтобы не было необходимости снова печатать весь документ.

6.1.1. Инициализация порта принтера/повторная инициализация принтера

Программы должны инициализировать порт каждого принтера (LPT1 - LPT3) перед первым его использованием, а также повторно инициализировать после устранения причин ошибки принтера. Не путайте инициализацию порта принтера с инициализацией самого принтера. Инициализация принтера это внутреннее дело принтера. Она происходит автоматически при его включении и в большинстве случаев принтер не может быть повторно инициализирован без его выключения и повторного включения. Но программа может повторно инициализировать включенный принтер, т.е. могут быть восстановлены начальные параметры работы принтера, отменяя все специальные шрифты, остановы табуляции и т.д. Считается правилом хорошего тона производить такой сброс принтера, после того как программа завершает работу с ним.

Языки высокого уровня инициализируют порт принтера автоматически, но программы на языке ассемблера требуют для этой цели короткой процедуры. Однако восстановление начальных параметров печати требуется во всех программах. Некоторые принтеры, такие, как новые эпсоновские принтеры, имеют "главный код сброса", который приводит к полному сбросу принтера. Но поскольку не все принтеры имеют такой код, программа должна предусматривать в своей завершающей части восстановление всех измененных параметров. Например, она может подать коды выключения курсива, плотной печати и т.д. Не забудьте включить вызов этой процедуры в процедуру выхода по «Ctrl-Break».

Помните, что на многих принтерах символы не печатаются до тех пор, пока не получен код возврата каретки, завершающий строку (или до тех пор, пока не введена целая строка данных). Символы могут спокойно ожидать в буфере принтера, даже после того, как породившая их программа завершилась. Когда начинается новая передача данных на принтер, эти символы будут напечатаны. Чтобы избежать этой проблемы, не забывайте почистить буфер перед началом печати; а также возьмите себе за правило чистить буфер при завершении программы. Это делается посылкой на принтер кода ASCII 24 (при этом параметры печати не меняются).

Средний уровень

Функция 1 прерывания 17H BIOS инициализирует порт принтера и возвращает байт, выдающий статус порта. Поместите в DX номер порта - число от 0 до 2 для LPT1 - LPT3, после чего вызовите прерывание. Байт статуса принтера (идентичный обсуждаемому в [6.1.2]) возвращается в AH.

```
;---инициализация LPT1
MOV  AH,1          ;функция инициализации принтера
MOV  DX,0          ;LPT1
INT  17H           ;проводим инициализацию
```

Низкий уровень

Регистр управления выводом каждого адаптера принтера имеет бит, который вызывает инициализацию адаптера. Этот регистр имеет адрес порта на 2 больше, чем базовый адрес адаптера. Напоминаем, что базовый адрес для LPT1 хранится в ячейке 0040:0008, для LPT2 - в 0040:000A и т.д. Имеют значение только младшие 5 бит регистра управления выводом. Бит 2 - бит инициализации принтера и обычно он устанавливается в 1. Для инициализации адаптера надо сбросить этот бит в 0 на тысячу тактов пустого цикла (3000 для AT или на 1/20 секунды, если используется счетчик времени суток BIOS [2.1.5]). В этот момент нужно, чтобы был установлен только бит 3 (принтер выбран). Поэтому пошлите в порт значение 12, сделайте задержку, а затем пошлите в порт обычное (без прерываний) инициализирующее значение, равное 8.

В следующем примере инициализируется LPT1:

```
;---инициализируем LPT1
        MOV DX,ES:[8]  ;считываем базовый адрес в DX
        INC  DX        ;прибавляем 2 к базовому адресу
        INC  DX
        MOV AL,12      ;значение для инициализации
        OUT DX,AL      ;начинаем инициализацию
DELAY:  MOV AX,1000    ;начало пустого цикла
        DEC  AX        ;уменьшаем счетчик
        JNZ  DELAY     ;повторяем 1000 раз
        MOV AL,8       ;обычное значение для регистра
        OUT DX,AL      ;конец инициализации
```

6.1.2. Проверка того, что принтер связан с машиной

Перед тем как послать на него вывод, программа всегда должна проверить, что принтер связан с машиной. Легко выявить, что принтер не готов, так как бит 3 регистра статуса принтера в этом случае устанавливается в 1. Но намного сложнее точно определить, почему принтер не готов: выключен ли он, отменен ли выбор принтера или в нем нет бумаги. Это происходит из-за того, что принтеры разных производителей посылают разные наборы битов в регистр статуса принтера, даже когда они находятся в идентичном состоянии. Хотя регистр статуса имеет биты, которые должны показывать все три состояния принтера, но в реальности значения битов могут не соответствовать этим условиям (бит 3 должен показывать, что принтер выключен, бит 4, что отменен выбор принтера, и бит 5, что нет бумаги). Приведенные ниже значения возвращаются в регистр статуса соответственно стандарту "Эпсон", которому обычно следует IBM:

Значение	Цепочка битов	Интерпретация
223	11011111	принтер готов
87	01010111	принтер не готов
119	01110111	нет бумаги в принтере
247	11110111	принтер выключен

Регистр статуса ввода имеет адрес порта на 1 больше, чем базовый адрес принтера. Базовый адрес для LPT1 хранится по адресу 0040:0008, для LPT2 - по адресу 0040:000A и т.д. Имейте в виду, что если принтер был выключен, то ему требуется некоторое время на инициализацию после включения. Не начинайте печатать до тех пор, пока регистр статуса ввода не сообщит, что принтер связан с машиной и готов к приему данных.

Высокий уровень

Данная процедура проверяет, связан ли принтер с машиной, и объясняет пользователю, что делать, если нет. Она использует значения из приведенной выше таблицы. Как уже отмечалось, такой подход не приемлем для процедуры общего назначения, которая будет обслуживать множество разных принтеров, но он вполне применим, когда вы пишете драйвер данного печатающего устройства. Отметим, что в строке 120 вычисляется двухбайтовое число, путем умножения старшего байта на 256 и добавления результата к младшему байту. Для нахождения адреса регистра

статуса ввода к значению полученного базового адреса добавляется 1.

```
100 '''Получаем адрес LPT1 и проверяем, готов ли принтер
110 DEF SEG = &H40                          'указываем на область BIOS
120 PB = PEEK(9) + 256*PEEK(8) + 1          'адрес регистра статуса
130 IF INP(PB) = 223 THEN 180               'если принтер готов
140 BEEP                                    'иначе - звонок и проверки
150 IF INP(PB) = 87 THEN locate 1,1: print"Strike the SELECT key" GOTO 150
160 IF INP(PB) = 247 THEN locate 1,1: print"Turn the printer on": GOTO 160
170 IF INP(PB) <> 223 THEN 170              'ждем инициализации
180 '''Теперь принтер on-line -- можно начинать печать
190 LPRINT Z$
```

Средний уровень

Для получения байта статуса из порта принтера используйте функцию 2 прерывания 17H. При входе DX содержит номер LPT (0-2 для LPT1-LPT3). Эта функция сбрасывает три неиспользуемых бита байта и делает операцию исключающего ИЛИ над двумя другими, поэтому значения битов отличаются от приведенных выше:

Значение	Цепочка битов	Интерпретация
144	10010000	принтер готов
24	00011000	принтер не готов
184	10111000	принтер выключен

И опять необходимо помнить, что эти значения меняются в зависимости от принтера. Наиболее общую информацию "выключен или не готов" дает бит 3 статуса, равный 0.

Низкий уровень

В следующем примере демонстрируется наиболее простая процедура - проверка бита on-line регистра статуса. Для получения байта статуса используется базовый адрес LPT1.

```
;---в сегменте данных
MES   DB    'Printer not ready - strike any key when OK$'
;---проверка, связан ли принтер с машиной (on-line)
      MOV   AX,40H      ;ES указывает на область данных BIOS
```

```
        MOV  ES,AX
        MOV  DX,ES:[8]    ;получаем базовый адрес
        INC  DX           ;смещение для регистра статуса
        IN   AL,DX        ;получаем байт статуса в AL
        TEST AL,1000B     ;проверяем бит 3
        JNZ  GO_AHEAD ;если принтер on-line, то вперед
;---печатаем сообщение об ошибке и ждем нажатия клавиши
        MOV  AH,9         ;функция вывода строки
        LEA  DX,MES       ;DS:DX указывают на сообщение
        INT  21H          ;печатаем сообщение
        MOV  AH,7         ;функция ожидания ввода
        INT  21H          ;ожидаем нажатия клавиши (без эха)
GO_AHEAD:                 ;продолжение программы
```

6.1.3. Интерпретация ошибок принтера и восстановление после них

Проверка на ошибки не должна завершаться на том, что вы убедились, что принтер связан с машиной. Ошибки принтера могут происходить в любой момент печати и программа должна быть готова восстановить ситуацию при сбоях. Хотя на принтере могут происходить самые разнообразные ошибки, только три типа ошибок возвращают информацию о себе в компьютер. Это - ошибка "отсутствия бумаги", ошибка "отсутствия связи с машиной" и общее сообщение "произошла ошибка". Как уже говорилось в [6.1.2], не все принтеры сообщают об этих ошибках одинаково, но теоретически регистр статуса ввода использует следующие биты:

бит	3 = 0	когда произошла ошибка на принтере
	4 = 0	когда принтер не связан с машиной (off-line)
	5 = 1	когда кончилась бумага на принтере

В частности, бит 4 может иметь другое назначение, нежели указано выше. Регистр статуса ввода имеет адрес порта, который на 1 больше, чем базовый адрес принтера. Базовый адрес для LPT1 хранится по адресу 0040:0008, для LPT2 - по адресу 0040:000A и т.д.

На низком уровне, когда программа посылает данные на принтер, она постоянно обращается к биту 7 этого регистра, чтобы проверить, готов ли принтер принять очередной символ. Не сложно при этом проверить и бит 3, чтобы узнать о произошедшей ошибке. Если происходит ошибка, индицируемая битами 4 и 5, то

по меньшей мере бит 3 будет равен 0. Программа должна постараться проанализировать ошибку, а затем попросить пользователя исправить ситуацию. Отметим, что функцию DOS, которая выводит символы на принтер (функция номер 5 прерывания 21H, см. [6.3.1]), можно заставить непрерывно проверять принтер на ошибку тайм-аута посредством команды MODE. Перед загрузкой программы, использующей функцию 5, надо ввести команду MODE LPT1: ,,P (еще лучше поместить эту команду в файл AUTOEXEC.BAT, с тем чтобы она всегда выполнялась при загрузке системы).

Все эти ошибки приводят к тому, что печать останавливается и должны быть предприняты какие-то действия, прежде чем она будет продолжена. Слишком огорчительно, если при возникновении ошибки на принтере, пользователю программы придется печатать заново большую порцию документа. Тщательно продуманная процедура восстановления после ошибки позволит программе возобновить печать с начала той страницы, на которой произошла ошибка. Приступая к печати новой страницы, необходимо всегда запоминать местоположение указателя выводимых данных. Процедура восстановления может попросить пользователя вставить новый лист бумаги, а затем, как уже отмечалось, продолжить печать с начала той страницы, на которой произошла ошибка.

Высокий уровень

В Бейсике распознаются два кода ошибки для принтера. Код ошибки 24 возвращается, если был отменен выбор принтера, а код 27 - если принтер выключен или в нем отсутствует бумага. Эти коды можно получить с помощью техники обнаружения ошибок, приведенной в [7.2.5]. К сожалению, эффективно отлавливается только код 27. Чтобы зарегистрировать код 24, требуется примерно полминуты, в течение которых программа заморожена. Не решит проблему и непосредственное чтение регистра статуса перед каждой операцией печати. Этот метод сработает перед началом печати, но ничем не поможет, если во время печати произойдет отмена выбора принтера. Приведем процедуру обработки ошибок принтера:

```
100 ON ERROR GOTO 1000                    'устанавливаем обработку ошибок
.
.
1000 '''проверяем, произошла ли ошибка на принтере
1010 IF ERR = 24 OR IF ERR = 27 THEN GOSUB 2000: RESUME
                                          'это ошибка принтера?
```

```
.
.
2000 BEEP: LOCATE 1,1: PRINT"Printer not ready" 'сообщаем пользователю
2010 PRINT "Strike any key when ready"  'о проблеме
2020 IF INKEY$ = "" THEN 2020          'ожидаем ввода
2030 RETURN                            'возвращаемся в программу
```

Средний уровень

Когда функция 0 прерывания 17H выводит символ на принтер, она возвращает байт статуса принтера в AH. Проверяйте значение этого байта после посылки каждого символа. BIOS слегка модифицирует байт статуса. Обычно бит 0 не имеет значения, но в данном случае он устанавливается, когда происходит ошибка тайм-аута (принтер не связан с машиной). В следующем примере проверяются два типа ошибок: общая ошибка "принтер не готов" и ошибка "отсутствие бумаги". В примере предполагается, что в начале каждой страницы (т.е. после каждого перевода формата) программа запоминает указатель начала выводимых данных, помещая его в переменную STARTING_PTR. Это позволяет программе при возникновении ошибки повторить печать с начала страницы, а не с начала всего документа. Конечно, принтер необходимо инициализировать еще раз перед повторной печатью, так чтобы были восстановлены все его параметры. (Данный пример просто иллюстрирует проверку ошибок - он ни в коей мере не является рабочей процедурой.)

```
;---в сегменте данных
MES1   DB   'Printer off-line - strike any key when ready$'
MES2   DB   'Printer out of paper - strike any key when ready$'
;---посылаем символ и проверяем на ошибку
NEXTC:MOV AH,0                  ;номер функции
      MOV DX,0                  ;выбираем LPT1
      MOV AL,[BX]               ;BX указывает на данные
      INC BX                    ;увеличиваем указатель
      INT 17H                   ;посылаем символ на принтер
      TEST AH,00001000B         ;выделяем бит 3 (флаг ошибки)
      JZ   NEXTC                ;если нет ошибки, то печатаем дальше
      TEST AH,00100000B         ;выделяем бит 5 (отсутствие бумаги)
      JZ   OFFL                 ;переход, если с бумагой все в порядке
      MOV AH,9                  ;готовим печать сообщения
```

```
        LEA  DX,MES2              ;DS:DX указывают на строку
        INT  21H                  ;выводим строку
        JMP  SHORT REC            ;уходим на восстановление
OFFL:   MOV  AH,9                 ;готовим печать сообщения
        LEA  DX,MES1              ;DS:DX указывают на строку
        INT  21H                  ;выводим строку
REC:    MOV  BX,STARTING_PTR      ;восстанавливаем указатель
        MOV  AH,0                 ;функция ожидания ввода
        INT  16H                  ;ждем
        CALL PRTR_INIT            ;инициализация принтера
        JMP  NEXTC                ;начинаем печать с начала страницы
```

6.1.4. Переключение между двумя или несколькими принтерами

Компьютеры, оснащенные несколькими параллельными портами, могут иметь одновременно подсоединенными два или более принтера. Вывод может перенаправляться с одного принтера на другой двумя способами. Один способ состоит в том, чтобы использовать только такие операторы вывода на печать, которые указывают, на какой принтер надо осуществлять вывод. Вы можете написать такой код, который позволит вам изменять спецификацию.

Во втором способе переключения принтеров вывод на LPT1 происходит по умолчанию, но с указанием другого принтера, который будет использоваться в качестве LPT1, что достигается изменением базового адреса, относящегося к LPT1. Этот базовый адрес хранится в области данных BIOS в ячейке 0040:0008. Поменяйте его с базовым адресом для LPT2 или 3 (хранящимися в ячейках 0040:000A и 0040:000C) и в качестве LPT1 будет использоваться другой адаптер.

Высокий уровень

В Бейсике, если принтер был открыт оператором OPEN "LPT1" AS 1, то, чтобы переключиться на другой принтер, надо сначала написать оператор CLOSE #1, а затем открыть другой принтер с помощью оператора OPEN "LPT2" AS #1. Впоследствии все операторы PRINT #1 будут направлять свой вывод на второй принтер. Это изменение труднее осуществить в программах, использующих оператор LPRINT, поскольку LPRINT по умолчанию посылает весь вывод на LPT1. В этом случае вам необ-

ходимо поменять базовые адреса принтеров. Именно это и делает следующая программа на Бейсике, переключая LPT1 и LPT2. Ее повторное использование переключает адреса обратно, возвращая систему к первоначальной конфигурации.

```
100 DEF SEG = &H40      'указываем на область данных BIOS
110 X = PEEK(8)         'получаем младший байт адреса LPT1
120 Y = PEEK(9)         'получаем старший байт адреса LPT1
130 POKE 8,PEEK(10)     'переносим младший байт адреса LPT2
140 POKE 9,PEEK(11)     'переносим старший байт адреса LPT2
150 POKE 10,X           'посылаем младший байт LPT1 в LPT2
160 POKE 11,Y           'посылаем старший байт LPT1 в LPT2
170 SYSTEM              'выходим из Бейсика
```

Эта программа будет очень кстати, если готовое программное обеспечение не адресуется к нужному принтеру. Ее можно откомпилировать и хранить на диске, скажем, под именем OTHERPRN, после чего надо будет только напечатать ее имя (в ответ на запрос DOS), чтобы переключиться с принтера на принтер. Если у вас нет транслятора с Бейсика, то создайте командный файл OTHERPRN.BAT и поместите в него строку BASIC OTHERPRN. Когда вы напечатаете OTHERPRN, будет автоматически загружен Бейсик, который загрузит и выполнит программу OTHERPRN.BAS, после чего вы вернетесь в операционную систему. Необходимо, правда, чтобы на диске имелся интерпретатор Бейсика BASIC.COM. Помните, что вы должны устоять перед искушением испытать эту программу до того, как она будет записана на диск, поскольку если вы ее запустите, она сотрет себя.

Низкий уровень

Один из способов, с помощью которого программа на ассемблере может переназначить принтер для вывода, состоит в использовании для печати функции 0 прерывания 17H [6.3.1]. Эта функция требует, чтобы номер принтера был помещен в DX. Заведите переменную для этого номера, с тем чтобы он мог быть изменен в любой момент. Вторая возможность состоит в обмене базовых адресов LPT1 и LPT2 или LPT3, что и демонстрирует следующая программа. Как и все короткие утилиты, она должна писаться в COM-форме, как объяснено в [1.3.6].

```
;---обмен базовыми адресами LPT1 и LPT2
MOV  AX,40H             ;сегмент области данных BIOS
```

```
MOV  ES,AX              ;ES указывает на данные
MOV  BX,8               ;смещение для базового адреса LPT1
MOV  DX,ES:[BX]         ;сохраняем базовый адрес LPT1
MOV  AX,ES:[BX] + 2     ;сохраняем базовый адрес LPT2
MOV  ES:[BX],AX         ;меняем базовый адрес LPT2
MOV  ES:[BX] + 2,DX     ;меняем базовый адрес LPT1
```

6.2. Установка спецификаций печати

Для установки различных спецификаций, относящихся к формату страницы, стилю шрифта и т.п., на принтер посылаются специальные управляющие коды. Эти коды посылаются на принтер так же, как и любые другие данные. Некоторые из них это простые однобайтовые коды из числа первых 32 набора кодов ASCII. Подобные управляющие коды, перечисленные в [7.1.9], инициируют такие простые действия принтера, как перевод строки или перевод формата (прогон страницы). Однако большинство спецификаций печати устанавливается посылкой Esc-последовательностей, в которых один или более кодовых байтов следуют за символом Esc, имеющего код - ASCII 27. Начальный код Esc информирует принтер, что символ(ы), который следует за ним необходимо интерпретировать как команду, а не как данные. Такие Esc-последовательности обычно не имеют символа-ограничителя, поскольку принтер "знает" длину каждой последовательности. Только в некоторых случаях, когда последовательность может иметь разную длину, требуется ограничивающий символ, в качестве которого всегда используется код ASCII 0.

Почти во всех случаях спецификации, установленные этими кодами, действуют до тех пор, пока они не будут явно отменены. Как только будет получен код, например, подчеркивания, то оно будет осуществляться до тех пор, пока не будет послан код отмены подчеркивания. Буфер принтера может быть очищен без отмены установленных спецификаций. Но если произошла ошибка на принтере и принтер был выключен и включен, то необходимо снова устанавливать все спецификации.

Большинство устанавливающих спецификации принтера кодов перемешано с данными, на которые они действуют. Например, данные для слова, которое должно быть выделено жирным шрифтом, должны предваряться Esc-последовательностью, включающей жирный шрифт, и завершаться Esc-последовательностью, выключающей его. Поскольку универсальный стандарт на эти

коды отсутствует, то печать с использованием мощных возможностей требует, чтобы для каждого поддерживаемого принтера были написаны драйверы. Каждый драйвер преобразует инструкции, генерируемые процедурой печати, в протокол, используемый данным принтером.

В ассемблере посылка кодов осуществляется самым обычным образом, но, работая в Бейсике, вы должны помнить, что операторы, посылающие управляющие коды (LPRINT или PRINT#), должны завершаться точкой с запятой. В противном случае операторы будут автоматически добавлять к посылаемым кодам пару возврат каретки/перевод строки.

Приведенные ниже обсуждения и примеры в основном относятся к графическому принтеру IBM. Коды, используемые этим принтером, настолько же "стандартны", насколько и любой другой протокол. В большей степени это связано с тем, что этот протокол применяется в эпсоновских принтерах (первые принтеры для IBM PC поставлялись фирмой Epson), которые составляют треть всех существующих принтеров. Управляющие коды, используемые принтерами IBM, сравниваются в [6.2.7]. Хотя информация, приведенная в данном разделе, может быть неприменима к тому принтеру, с которым вы работаете, но все-таки большинство общих принципов окажутся полезными вам.

6.2.1. Установка текстового и графического режимов

Пока принтер специально не переведен в графический режим, он всегда находится в текстовом режиме. Команда, устанавливающая графический режим, должна сообщать, какое число байтов графических данных будет передано (но не больше одной строки), и после того, как это число байтов будет интерпретировано как графическое изображение, принтер вернется в текстовый режим. По этой причине нет команды, которая переводит принтер в текстовый режим.

Число графических режимов у разных принтеров различно. Во всех случаях за кодом, устанавливающим графический режим, следуют 2 байта, указывающие, какое число графических байтов будет передано (сначала младший байт). Чтобы вычислить значение этих двух байтов, разделите число байтов данных на 256 и поместите результат во второй байт, а остаток - в первый байт. За этими двумя байтами должны сразу следовать байты данных.

Каждый байт определяет цепочку битов, соответствующих восьми вертикальным точкам одной позиции в строке. Младший бит (1) соответствует низу колонки, а старший бит (128) - верху. Например, чтобы напечатать пирамиду, пошлите сначала байт, у

которого установлен только нижний бит, затем байт, у которого установлены 2 нижних бита, и т.д. После восьмого байта расположите те же байты в обратном порядке. Значение первого байта будет 1, второго - 3 (1 + 2), затем - 7 (1 + 2 + 4), 15 (1 + 2 + 4 + 8) и т.д. На рис. 6.1 изображена вся картина.

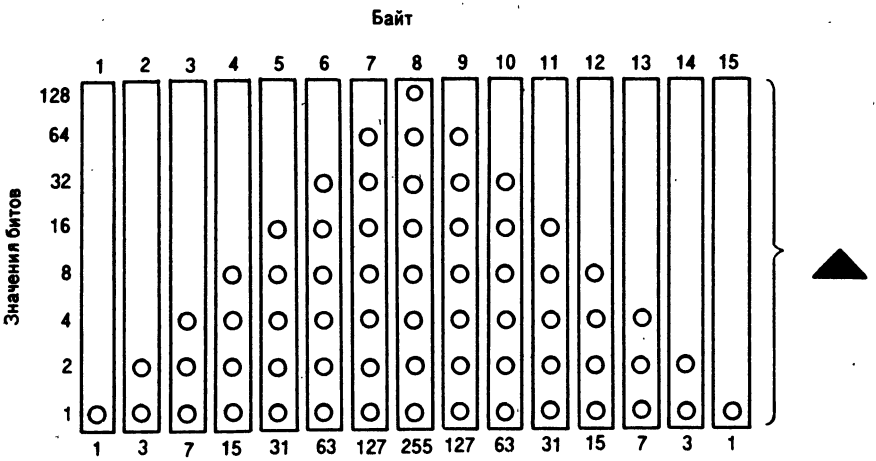


Рис. 6.1. Цепочка битов для печати пирамиды

Для печати пирамиды в Бейсике на графическом принтере IBM напишите следующий код:

```
100 LPRINT CHR$(27);CHR$(75);CHR$(15);CHR$(0);CHR$(1);CHR$(3);
CHR$(7);CHR$(15);CHR$(31);CHR$(63);CHR$(127);CHR$(255);
CHR$(127);CHR$(63);CHR$(31);CHR$(15);CHR$(3);CHR$(1);
```

Первые два байта переводят принтер в графический режим с 480 точками, следующие два - сообщают, что будет передано 15 байт графических данных, а затем идет последовательность байтов данных. Конечно, то же самое можно запрограммировать умнее, организовав цикл, в котором будут передаваться байты данных. Отметим, что в этом случае проблемы возникнут, если указанное число байтов не соответствует числу посылаемых байтов. Чтобы создать пробел между графическими фигурами, выведите несколько байтов с нулевым значением. В Бейсике, когда в одной строке выводится больше 80 байт графических данных, не

забудьте предварительно установить "бесконечную" ширину принтера. Для этого надо ввести команду WIDTH "LPT1:",255.

Графический принтер IBM имеет четыре графических режима, которые более или менее "стандартны". Они таковы:

27,75 480 точек в строке. Нормальный режим. Максимум 480 байт данных на оператор.

27,76 960 точек в строке. Удвоенное горизонтальное разрешение, но печать вдвое медленнее (двойная плотность). Максимум 960 байт данных на оператор.

27,89 960 точек в строке, печать с нормальной скоростью (двойная плотность с высокой скоростью). Две точки, прилегающие по горизонтали, не могут быть напечатаны, поскольку не будут успевать иголки печатающей головки. Если делается попытка их напечатать, то вторая точка будет игнорироваться. Максимум 960 байт данных на оператор.

27,90 1920 точек в строке, печать вдвое медленнее (четверная плотность). Соседние точки по горизонтали должны отстоять по крайней мере на 3 точки (т.е. 1 печатаем, 2 пропускаем). Максимум 1920 байт данных на оператор.

В более плотных режимах две прилегающие по горизонтали точки не могут быть напечатаны. Чтобы заполнить пропуски между точками, верните каретку к левому полю, немного сдвиньте печатающую головку вправо и сделайте второй проход, используя те же данные. Сравним плотности печати, вызываемые одними и теми же управляющими кодами на разных принтерах:

Коды	Графический принтер	Цветной принтер	Компактный принтер	Проприятер
27,75	480 точек	1108	560	480
27,76	960 точек	2216	-	960
27,89	960 точек	2216	-	960
27,90	1920 точек	4432	-	1920

Из всех принтеров фирмы IBM цветной принтер является уникальным, так как он может устанавливать масштабный коэффициент (aspect ratio) для графических изображений. Этот коэффициент отражает разницу горизонтальных и вертикальных расстояний между точками. Обычно желателен коэффициент 1:1, поскольку в противном случае трудно проводить графические вычис-

ления. Но при копировании графического изображения с экрана надо, чтобы масштабный коэффициент был таким же, как у дисплея. В экранном режиме умеренного разрешения 5 точек по вертикали занимают тот же размер, что и 6 точек по горизонтали. Это соответствует масштабному коэффициенту 5:6, каковое и используется по умолчанию цветным принтером. Допускаются только коэффициенты 1:1 и 5:6.

6.2.2. Управление расстоянием между строками

В принтерах, за исключением тех, в которых заложены возможности графопостроителя, вся печать осуществляется построчно. Даже графические изображения рисуются построчно, хотя в этом случае нет пустых мест между строками. Код ASCII 10 - стандартный управляющий код перевода строки. Посылка его на принтер (без предшествующего кода Esc) приводит к тому, что бумага будет продвинута вперед на указанный интервал. Обычно, если код перевода строки не посылается за кодом возврата каретки, печатающая головка возвращается к левому краю бумаги и можно снова печатать на той же строке. Однако можно сделать и так, чтобы перевод строки происходил автоматически при каждом возврате каретки. Этим управляют переключатели на принтере. Того же результата можно добиться, установив бит 1 регистра управления выводом (см. [6.1.0]). Многие принтеры выполняют включение и выключение автоматического перевода строки с помощью управляющих кодов 27,53, а некоторые - осуществляют обратный перевод строки с помощью кодов 27,93.

По умолчанию графический принтер использует интервал печати, равный 1/6 дюйма (т.е. выводят 6 строк на дюйм); к этому режиму всегда можно вернуться, послав управляющие коды 27,50 (эти коды используются также в сочетании с кодами изменения интервала между строками, обсуждаемыми ниже). Для графического принтера имеются еще два предопределенных межстрочных интервала: 1/8 дюйма и 7/72 дюйма. Соответствующие им управляющие коды 27,48 и 27,49.

Возможна и более тонкая градация межстрочных интервалов. В графическом принтере применяются три кода, позволяющие изменить интервал на очень малую величину. Все три управляющих кода используют двухбайтовую Esc-последовательность, за которой следует число 1/72 или 1/216 дюйма, определяющее межстрочный интервал. Вертикальное расстояние между центрами двух точек равно 1/72 дюйма. Интервал 8/72 дюйма не оставляет промежутка между строками (9 строк на дюйм). Стандартный интервал 6 строк на дюйм задается числом, равным 12/72 дюйма.

Наконец, 1/216 соответствует 1/3 от 1/72. Изменение на такую величину позволяет печатающей головке слегка сдвинуться от центра строки, с тем чтобы точки при втором проходе заполнили промежутки, обеспечивая печать более высокого качества. Ниже приводятся Esc-последовательности для управляющих кодов:

Изменение	Esc-последовательность
1/72 дюйма	27,65,n (где n от 1 до 85)
1/216 дюйма	27,51,n (где n от 1 до 255)
1/216 дюйма	27,74,n (где n от 1 до 255)

Команды для изменения интервала, выраженного в 1/72 дюйма, не активизируются до тех пор, пока не встретится второй управляющий код: 27,50. Как объяснялось выше, этот код может также использоваться отдельно для восстановления стандартного интервала в 1/6 дюйма. Если ранее была применена команда 27,65,n, то для восстановления интервала в 1/6 дюйма надо послать команду 27,65,12,27,50. Два управляющих кода для интервалов, выраженных в 1/216 дюйма, не идентичны. Первый код определяет, что все последующие переводы строки будут выполняться с указанным интервалом; второй же действует только на один перевод строки, а затем возвращает интервал, установленный ранее.

Следующая таблица сравнивает межстрочные интервалы, вызываемые одними и теми же управляющими кодами на различных принтерах фирмы IBM:

Коды	Матричный	Графический	Цветной	Компактный	Струйный	"Ромашковый"	Проприntер
27,48	1/8	1/8	1/8	1/9	1/8	1/8	1/8
27,49	7/72	7/72	6/72	1/9	9/96	7/72	
27,50	1/6	1/6	1/6	1/6	1/6	1/6	1/6
27,51		n/216	n/144				n/216
27,65	n/72	n/72	n/72			n/72	n/72
27,74		n/216	n/144				n/216

Независимо от того, как изменяются межстрочные интервалы, принтер всегда контролирует прямые и обратные движения листа, поэтому переход к началу следующей страницы делается всегда вовремя.

6.2.3. Управление движением бумаги

Бумага в принтере передвигается с помощью команд перевода строки, вертикальной табуляции и перевода формата. Установкой переключателей на принтере определяется, будет ли принтер автоматически переходить на новую страницу при обнаружении перфорации между страницами. Если перфорация не будет пропускаться, то печать может завершиться прямо на верхнем краю очередной страницы. Пропуск перфорации оставляет по три пустых строки сверху и снизу каждой страницы. На самом деле принтер не распознает перфорацию, вместо этого он, предполагая, что в начальный момент бумага выравнена на начало страницы, считает число переводов строки. Можно программно переопределить установку переключателей, посылая на принтер управляющие коды 27,56, чтобы он не делал пропуска перфорации, и 27,57, чтобы делал пропуск перфорации.

Графический принтер использует код, который определяет число строк, пропускаемых между страницами. Этот код 27,78,n, где n - число строк от 1 до 127. Например, применение кода 27,78,10 приведет к тому, что принтер будет пропускать по 10 строк. Если межстрочный интервал равен 1/6 дюйма, то 11-дюймовая страница будет содержать 66 строк и после печати каждых 56 строк принтер будет пропускать 10 строк. Ваша же программа должна позаботиться, чтобы в самом начале продвинуть бумагу на 5 строк, с тем чтобы 55 строк текста помещались в центре страницы.

Если используется бумага, размер которой отличается от стандартного 11-дюймового, то можно изменить длину страницы, с тем чтобы пропуски перфорации происходили в нужном месте и чтобы перевод формата устанавливал бумагу в правильную позицию. Размер страницы может устанавливаться либо числом строк на странице, либо размером в дюймах. Чтобы установить число строк на странице, пошлите код 27,67,n, где n - число строк. Та же последовательность используется и для установки длины страницы в дюймах, за исключением того, что длина страницы записывается в форме 0,n, где n может быть от 1 до 22 дюймов. Для стандартной страницы надо послать команду 27,67,0,11.

6.2.4. Управление положением печатающей головки

Печатаемый текст распределяется по странице частично за счет движения бумаги [6.2.3], а частично за счет движения печатающей головки. Головка может быть позиционирована в любое место, но не путем задания ее координат. Вместо этого

указывается ее смещение относительно самой левой позиции, которую она может занимать. У принтера нет датчиков, сообщающих текущее положение головки. Ваша программа при необходимости должна отслеживать положение головки. При этом считается разумным начинать печать с подачи управляющего кода 27,60, который сдвигает головку в самую левую позицию, не делая перевода строки (то же самое делает и код возврата каретки).

При печати текста имеется несколько способов передвинуть головку в нужное положение. Она может сдвигаться вправо подачей одного или нескольких символов пробела или табуляции и влево подачей одного или нескольких символов "возврат на шаг" или символа возврата каретки. Движения осуществляются непрерывно - не стоит воспринимать их как нечто похожее на действия обычной пишущей машинки. До тех пор пока ваша программа знает начальное положение печатающей головки, она может комбинацией переводов строки, пробелов, табуляций и возвратов на шаг форматировать вывод в соответствии с вашими пожеланиями. Принтеры, которые умеют выполнять обратный перевод строки, могут использоваться и как графопостроители.

В графических режимах возможно перемещение головки на малые доли дюйма. При печати текста вы можете войти в графический режим, чтобы добиться разных промежутков между словами. К сожалению, этот процесс существенно замедляет печать. Смотрите пример в [6.3.2].

Имеется специальный код, который заставляет головку всегда возвращаться в крайнюю левую позицию перед печатью очередной строки, отменяя двунаправленную печать. Хотя это значительно замедляет печать, однако при этом достигается более точное позиционирование головки. Это особенно полезно при работе в графическом режиме. Для того чтобы включить однонаправленную печать, надо послать код 27,85,1, а чтобы вернуться к двунаправленной печати - код 27,85,0.

6.2.5. Установка позиций табуляции

В зависимости от принтера могут устанавливаться позиции горизонтальной и вертикальной табуляции (графический принтер IBM не имеет вертикальной табуляции). Под горизонтальной табуляцией понимаются смещения относительно левого края, выраженные в пробелах. В некоторых случаях допускаются до 112 позиций горизонтальной табуляции. Аналогично вертикальные табуляции определяются как смещения относительно верха страницы, а измеряются они в межстрочных интервалах. Для большинства

принтеров IBM допускается не больше 64 позиций вертикальных табуляций.

Первые два байта кода для установки горизонтальной табуляции - 27,68, а для установки вертикальной табуляции - 27,66. Для обоих типов табуляций далее идет строка байтов, показывающая позиции табуляции в возрастающем порядке. Эта строка должна завершаться байтом ASCII 0, который служит ограничителем. Для установки горизонтальной табуляции в позициях 15, 30 и 60 пошлите на принтер код 27, 68, 15, 30, 60, 0. Для установки вертикальной табуляции в строках 8 и 12 пошлите код 27, 66, 8, 12, 0. Напомним, что если размер страницы отличается от стандартного, то он должен быть установлен перед определением позиций вертикальной табуляции. Вертикальная табуляция отменяется кодом 27,67.

Отметим, что большинство принтеров не имеет установки полей как таковой. Левое поле может создаваться за счет вывода табуляции или ряда пробелов в начале каждой строки. Для точной установки полей перейдите в графический режим и выведите ряд байтов ASCII 0. Правое поле создается путем ограничения длины строки.

6.2.6. Изменение шрифта печати

Ширина страницы 8 1/2 дюйма позволяет напечатать в строке до 80 обычных символов, если все они имеют одинаковую ширину. Пропорциональная печать (см. [6.3.3]) позволяет поместить в строке еще несколько символов. С другой стороны, плотная печать позволяет вывести в строке 132 символа, печать с двойной шириной - 40 символов, а плотная печать с двойной шириной - 64 символа. Имейте в виду, что использование печати с разной шириной в одной строке приведет к трудностям при форматировании.

Большинство матричных принтеров имеет набор режимов печати специальными шрифтами. Приведем перечень стандартных возможностей, предоставляемых графическим принтером IBM.

Плотная печать. Для включения режима плотной печати надо послать однобайтовый управляющий код 15, для выключения режима - код 18. Стандартная страница шириной 8 1/2 дюйма позволяет напечатать в этом режиме 132 символа в строке.

Печать с двойной шириной. Для того чтобы принтер начал печатать с двойной шириной, надо послать на него управляющий код 14. Режим печати с двойной шириной необычен тем, что принтер автоматически выключает этот режим, когда встречает символ

возврата каретки или перевода строки. Поскольку такой вид печати обычно используется для однострочных заголовков, то это свойство удобно. Чтобы выключить этот режим в середине строки, пошлите код 20.

Выделенная печать. При выделенной печати каждый символ печатается два раза в одной и той же позиции. Точки становятся темнее, что и создает эффект выделения. Скорость печати при этом уменьшается вдвое. Для включения этого режима пошлите код 27,69, для выключения - 27,70.

Печать за два прохода. В режиме печати за два прохода бумага сдвигается на 1/216 дюйма перед вторым проходом печатающей головки. При этом получаются более заполненные буквы, которые к тому же выглядят ярче. Скорость печати уменьшается вдвое. Этот режим включается управляющим кодом 27,71, а выключается кодом 27,72.

Печать с подчеркиванием. Печать с подчеркиванием может выполняться двумя способами. Графический принтер имеет режим подчеркивания, в котором линейка печатается под каждым символом, включая пробелы. Для графического принтера фирмы IBM такой режим включается кодом 27,45,1, а выключается кодом 27,45,0. Принтеры, не имеющие режима подчеркивания, могут сделать линейки при втором проходе по той же строке, печатая символы подчеркивания (ASCII 95) в тех местах, где это нужно, и пробелы (ASCII 32) - во всех остальных позициях. Второй проход достигается тем, что после первого прохода подается код возврата каретки без кода перевода строки. Второй проход не мешает принтеру правильно подсчитывать строки при вычислении размера страницы.

Печать с верхними и нижними индексами. На графических принтерах текст с верхними и нижними индексами сжимается вертикально. Для печати верхнего индекса пошлите управляющий код 27,83,0, а для печати нижнего - 27,83,1. Можно прямо переходить от одних индексов к другим. Для выключения печати индексов с тем, чтобы принтер оказался на текущей строке, пошлите управляющий код 27,84.

Некоторые режимы не могут использоваться в комбинации с другими. Если вы хотите включить 4 режима одновременно, то проконсультируйтесь со следующей таблицей. В каждом из шести столбцов приведена допустимая комбинация.

Режим печати	1	2	3	4	5	6
Нормальный	X	X				
Сжатый			X	X		
Выделенный					X	X
За два прохода	X		X		X	
С индексами		X		X		X
С двойной шириной	X	X	X	X	X	X
С подчеркиванием	X	X	X	X	X	X

6.2.7. Сравнение возможностей принтеров IBM

В следующей таблице сравниваются управляющие коды для принтеров фирмы IBM. Не вся информация относительно кодов точна (обращайтесь к документации IBM), а в ряде случаев опущены уникальные коды. Предлагаемая таблица показывает диапазон возможностей принтеров, а также выделяет коды, которые можно считать стандартными. Отметим, что коды для первых четырех принтеров приведены в выпуске "Возможности и адаптеры" (Options and Adapters) из серии технических руководств, а коды для остальных принтеров приведены в сопровождающих их руководствах по эксплуатации.

Код	Функция	Принтер						
		Матричный	Графический	Цветной	Компактный	Струйный	Ромашковый	Построчный принтер
	Перемещение бумаги:							
10	перевод строки	X	X	X	X	X	X	X
11	вертикальная табуляция	X		X	X	X	X	X
12	перевод формата	X	X	X	X	X	X	X
13	возврат каретки	X	X	X	X	X	X	X
27,52	установка начала страницы			X			X	X
27,56	игнорировать отсутствие бумаги	X	X					
27,57	отмена игнорирования отсутствия бумаги	X	X					

Код	Функция	Принтер						
		Матричный	Графический	Цветной	Компактный	Струйный	Ромашковый	Прочий принтер
27,66	установка вертикальных табуляций	X		X	X	X		X
27,66	очистка вертикальных табуляций							X
27,88	установка пропуска перфорации		X	X	X	X		X
27,79	отмена пропуска перфорации		X	X	X	X		X
Перемещение печатающей головки:								
8	возврат на шаг			X		X	X	X
9	горизонтальная табуляция	X	X	X	X	X	X	X
27,60	сдвиг головки в левый край	X	X	X				
27,62	установка индекса горизонт. движения					X		
27,68	установка горизонтальной табуляции	X	X	X	X	X	X	X
27,68	очистка горизонтальной табуляции							X
27,77	автоматическое форматирование			X				
27,80	вкл./выкл. пропорциональной печати		X			X		
27,82	восстановление табуляций по умолчанию			X	X		X	X
27,85	вкл./выкл. однонаправленной печати		X	X				
27,88	установка левого/правого поля			X			X	
27,100	программируемый пробел			X				
27,101	программируемый возврат на шаг			X				

Код	Функция	Принтер						
		Матричный	Графический	Цветной	Компактный	Струйный	Ромашковый	Проп принтер
Межстрочные и межсимвольные интервалы:								
27,48	межстрочный интервал 1/8 дюйма	X	X	X		X	X	X
27,48	межстрочный интервал 1/9 дюйма				X			
27,48	межстрочный интервал 7/72 дюйма	X	X					
27,49	межстрочный интервал 7/72 дюйма							X
27,49	межстрочный интервал 9/96 дюйма						X	
27,49	межстрочный интервал 6/72 дюйма			X				
27,49	межстрочный интервал 1/9 дюйма				X			
27,50	начать программируемый перевод строки по 27,65	X	X	X				
27,50	межстрочный интервал 1/6 дюйма	X	X	X	X	X	X	X
27,51	программируемый перевод строки(n/216)		X					X
27,51	программируемый перевод строки(n/144)			X				
27,53	вкл./выкл. автоматического перевода строки			X	X	X	X	X
27,65	программируемый перевод строки (n/72)	X	X	X			X	X
27,67	установка длины страницы	X	X	X	X	X	X	X
27,74	программируемый перевод строки (n/216)		X					X
27,74	программируемый перевод строки (n/144)			X				

Код	Функция	Принтер						
		Матричный	Графический	Цветной	Компактный	Струйный	Ромашковый	Пропринтер
27,93	обратный перевод строки			X			X	
27,104	перевод на пол-строки вперед						X	
27,105	перевод на пол-строки назад						X	
Управление шрифтами:								
11	режим 15 символов на дюйм						X	
14	включение режима двойной ширины	X	X	X	X	X		X
15	включение плотной печати	X	X	X	X	X		X
18	выключение плотной печати	X	X		X	X		X
18	режим 10 символов на дюйм			X		X	X	
20	выключение режима двойной ширины	X	X	X	X	X		X
27,45	вкл./выкл. подчеркивания		X	X	X	X	X	X
27,58	режим 12 символов на дюйм			X			X	X
27,69	включение жирной печати	X	X	X				X
27,70	выключение жирной печати	X	X	X				X
27,71	включение печати в 2 прохода	X	X	X		X		X
27,72	выключение печати в 2 прохода	X	X	X		X		X
27,83	включение печати индексов		X	X		X	X	X

Код	Функция	Принтер: Матричный	Графический	Цветной	Компактный	Струйный	Ромашковый	Пропринтер
27,84	выключение печати индексов		X	X		X	X	X
27,87	вкл./выкл. печати с двойной шириной		X	X	X	X		X
27,91	включение цветного подчеркивания					X		
27,95	вкл./выкл. overscore							X
	Установка специальных шрифтов и цветов:							
27,54	выбор набора символов 2		X	X		X	X	X
27,55	выбор набора символов 1		X	X		X	X	X
27,61	загрузка шрифта						X	X
27,73	изменение качества печати			X		X		X
27,92	печатать управляющие символы			X			X	X
27,94	печатать все символы			X			X	X
27,97	сдвиг ленты в конце страницы			X				
27,98	выбор 4-й полосы ленты			X				
27,99	выбор 3-й полосы ленты			X				
27,109	выбор 2-й полосы ленты			X				
27,121	выбор 1-й полосы ленты			X				
	Графические режимы:							
27,75	установка режима 480 точек		X					X

Код	Функция	Принтер						
		Матричный	Графический	Цветной	Компактный	Струйный	Ромашковый	Пропринтер
27,75	установка режима 560 точек				X			
27,75	установка режима 1108 точек			X				
27,76	установка режима 960 точек		X					X
27,76	установка режима 2216 точек			X				
27,89	установка режима 960 точек с нормальной скоростью		X					X
27,89	установка режима 2216 точек			X				
27,90	установка режима 1920 точек		X					X
27,90	установка режима 4432 точек			X				
27,91	установка разрешения/цвета					X		
27,110	установка масштабного коэффициента			X				
Другие возможности:								
7	звонок	X	X	X			X	X
17	выбор принтера	X		X		X	X	X
19	отмена выбора принтера	X		X		X	X	
24	очистка буфера	X	X	X	X	X	X	X
27,81	отмена выбора указанного принтера			X				X

6.3. Посылка данных на принтер

Посылка данных на принтер тривиальна в языках высокого уровня, а для программирующего на языке ассемблера имеется ряд функций операционной системы, которые также делают задачу достаточно простой. Программирование на низком уровне требует больше работы, но зато предоставляет больше возможностей. Как правило, процедуры печати низкого уровня посылают символ на принтер, а затем постоянно проверяют регистр статуса ввода порта, к которому присоединен принтер. Следующий символ посылается только тогда, когда принтер сигнализирует, что он готов (принтер может не печатать символ сразу, а запасать его в своем буфере до тех пор, пока не будет получена целая строка символов для печати).

Кроме того, процедуры низкого уровня могут использовать прерывание принтера или могут имитировать действие этого прерывания. С помощью специального программирования можно сделать так, что принтер будет осуществлять прерывание процессора, когда он готов к приему следующего символа. Процедура обработки прерывания посылает следующий символ, после чего процессор может продолжать заниматься своими делами. Этот метод используется для *фоновой печати* (которую называют также *спулингом*). Поскольку физические перемещения деталей принтера намного медленнее, чем быстродействие электроники компьютера, то вывод символов на принтер занимает лишь малую долю процессорного времени. Применение прерывания позволяет использовать это время эффективно.

При посылке данных на принтер требуются сравнительно небольшие усилия, чтобы добиться очень сложного вывода. Все сложные картинки, которые может выводить принтер, можно получить за счет комбинирования текстовых и графических данных, а также многочисленных кодов управления принтером, обсуждавшихся ранее в этой главе. Комбинируя в одной строке текстовый и графический режимы, можно добиться выравнивания правого поля и пропорциональной печати. Кроме того, любой графический принтер может создавать специальные символы произвольного вида, а за счет аккуратного манипулирования надпечатки и межстрочного интервала могут выводиться любые символы псевдографики.

6.3.1. Вывод текстовых или графических данных на принтер

Процессор может быть занят только посылкой данных на принтер или печатать в фоновом режиме, используя прерывания принтера. Возможна и третья альтернатива, когда программа посылает символы на принтер через определенные интервалы, что можно рассматривать как "псевдопрерывание". Этот метод не так тесно координируется с работой принтера, как настоящее прерывание, но это несущественно, так как работа принтера не критична ко времени.

Независимо от того, как выводятся данные, каждый раз на принтер посылается только 1 байт данных. Языки высокого уровня предоставляют функции, которые, казалось бы, выводят сразу целые строки, однако на самом деле эти функции разбивают строки на отдельные символы. Обычно языки высокого уровня посылают на принтер пару возврат каретки/перевод строки в конце каждой строки, тогда как программы на ассемблере должны сами добавлять эту пару кодов. Из-за этого приходится немного больше программировать, но взамен вы получаете более совершенную программу, особенно в отношении проверки ошибок.

Высокий уровень

Для посылки данных на принтер Бейсик предоставляет операторы LPRINT и PRINT#. LPRINT не требует никакой подготовки, но для вывода оператором PRINT# вы должны предварительно открыть принтер в точности так же, как и файл, с помощью оператора OPEN "LPT1" AS 1 или OPEN "LPT3" AS #2. Оператор LPRINT всегда адресуется к LPT1, в то время как PRINT# может адресоваться к любому принтеру.

Пара возврат каретки/перевод строки автоматически добавляется в конце любого оператора LPRINT или PRINT#, если только он не завершается точкой с запятой. Для избежания ненужных переводов строки не забывайте завершать посылку любых управляющих кодов точкой с запятой. То же самое надо сделать, если вы хотите, чтобы строки текста печатались подряд, прилегая одна к другой. Однако имейте в виду, что многие принтеры не начинают печатать до тех пор, пока они не получат данные для целой строки. Это определяется либо символом возврат каретки, либо тем, что число переданных символов достигло 80 (или другого числа). Не забывайте послать завершающий код возврата каретки, чтобы вытолкнуть последнюю порцию символов из буфера принтера.

Принтер автоматически переходит на следующую строку по достижении конца предыдущей. По умолчанию размер строки принтера равен 80 символам, но у принтеров с широкой кареткой это значение может быть больше. Строки, выводимые в режимах плотной печати или печати двойной ширины, могут иметь различную длину. Для изменения номера столбца, по достижении которого головка принтера перейдет на следующую строку, необходимо установить ширину принтера командой WIDTH "LPT1",n, где n - требуемый номер столбца. Когда печатается строка, длина которой больше или равна ширине принтера, печатающая головка переходит на следующую строку, что эквивалентно выполнению кодов возврат каретки/перевод строки. Это означает, что в случае, когда длина строки равна ширине принтера, будут сделаны два перевода строки, если эта строка завершается, как обычно, парой возврат каретки/перевод строки.

При графической печати принтер обычно устанавливают на бесконечную ширину. Чтобы сделать это, надо подать команду установки ширины, равной 255, WIDTH "LPT1",255. Если вы забудете включить эту команду, то при выводе длинных последовательностей графических данных Бейсик будет вставлять пару возврат каретки/перевод строки после каждых 80 байт данных. Эти добавочные символы будут включаться в общее число байтов данных для графической печати, поэтому конец передаваемых данных будет просто опущен.

Один оператор LPRINT может включать несколько элементов данных в различных видах. Информация может содержаться в самом операторе, как, например, в LPRINT "The rain in Spain", или на нее можно ссылаться по имени переменной, как в случае X$ = "The rain in Spain": LPRINT X$. Специальные символы могут включаться за счет использования функции CHR$. Управляющие коды обычно посылаются именно этим способом, например, LPRINT CHR$(10) посылает на принтер управляющий код перевода строки. Чаще всего CHR$ применяют при посылке кодов ASCII, которые нельзя ввести с клавиатуры. Любые из перечисленных типов данных могут быть объединены в одном операторе. Если вы хотите, чтобы различные элементы данных печатались подряд, то разделяйте их точкой с запятой; в случае разделения их запятыми следующий элемент будет выводиться со следующей позиции табуляции. Это говорит о том, что оператор LPRINT форматирует печать так же, как это делает оператор PRINT при выводе на экран. Приведем несколько примеров:

```
100 LPRINT S$;" and ";Y$          'комбинация трех строк
110 LPRINT X, Y, Z                'вывод трех чисел
```

```
120 LPRINT "The total is "; X                комбинация строки и числа
130 lprint "The ";chr$(27);chr$(45);chr$(1);"real";chr$(27);chr$(45);chr$(0);" thing."
140                                          подчеркивание среднего слова
```

Оператор PRINT # может использовать те же типы данных, что и оператор LPRINT, и он также позволяет включать несколько элементов данных в один оператор и смешивать различные типы данных. Точки с запятой и запятые действуют в нем аналогичным образом. Предлагаемые примеры, эквивалентны приведенным выше:

```
100 OPEN "LPT1:" AS #2
110 PRINT #2,S$;" and ";Y$
120 PRINT #2,X, Y, Z
130 PRINT #2,"The total is "; X
140 PRINT #2,"The ";chr$(27);chr$(45);chr$(1);"real";
chr$(27);chr$(45);chr$(0);" thing."
```

Средний уровень

Функция 0 прерывания 17H посылает один символ на принтер. Поместите символ в AL, а номер принтера - в DX. При возврате AH будет содержать регистр статуса, который надо постоянно проверять для обнаружения ошибок. В [6.1.3] объясняется, как это делать. Для вывода потока данных установите указатель на содержащий их буфер и напишите процедуру, подобную следующей:

```
;---вывод данных на LPT1
        MOV  CX,NUMC    ;CX содержит число байтов для вывода
        MOV  DX,0       ;выбираем LPT1
NEXTC:  MOV  AH,0       ;функция посылки символа на принтер
        MOV  AL,[BX]    ;BX указывает на буфер данных
        INT   17H       ;посылаем символ
        TEST AH,8       ;проверяем бит ошибки
        JNZ  PR_ERROR   ;уход на обработку ошибки
        INC  BX         ;увеличиваем указатель
        LOOP NEXTC      ;выводим следующий символ
```

Функция 5 прерывания 21H - стандартное прерывание MS-DOS для вывода на принтер. Поместите символ в DL и выполните прерывание. Эта функция всегда подразумевает вывод на LPT1 и у нее нет возвращаемых регистров.

```
;---вывод данных на LPT1
MOV  AH,5          ;номер функции
MOV  DL,CHAR       ;готовим печатаемый символ
INT  21H           ;посылаем его на принтер
```

Другой способ вывода данных на принтер - с помощью функции 40H прерывания 21H, которая является функцией стандартного вывода и применяет метод дескриптора файлов для доступа к файлу или устройству (см. [5.3.0]). При печати эта функция использует специальный предопределенный номер файла для принтера (номер #4), который надо поместить в BX. Функции доступен только LPT1, поэтому для вывода на другой принтер вам надо будет поменять базовые адреса (см. [6.1.4]). DS:DX должны указывать на выводимые данные, а CX - содержать число посылаемых байтов. Например:

```
;---вывод 120 байт данных на LPT1
MOV  AH,40H        ;номер функции
MOV  BX,4          ;номер файла для принтера
MOV  CX,120        ;число посылаемых байтов
LEA  DX,PRTR_DATA  ;DS:DX указывают на данные
INT  21H           ;посылаем данные
JC   PRTR_ERROR    ;на обработку ошибки
```

При возврате установленный флаг переноса индицирует ошибку, при этом AX будет содержать 5, если принтер не связан с машиной, и 6, если указан неверный номер файла. Отметим, что при использовании предопределенного номера файла открывать устройство не нужно.

Низкий уровень

Байт данных посылается на принтер, а именно в регистр выводимых данных, адрес порта которого совпадает с базовым адресом принтера. Помните, что базовые адреса для LPT1-LPT3 хранятся со смещениями 8, 10 и 12 в области данных BIOS (начинающейся с 0040:0000). После того как данные посланы в регистр, на короткое время включается бит строба регистра управления выводом, адрес порта которого на 2 больше, чем для регистра данных. Номер бита строба равен 0, и он должен быть установлен только на очень короткое время, чтобы инициировать передачу данных, находящихся в регистре данных. Процедура печати может немедленно сбросить бит строба обратно в 0.

После того как байт данных послан, программа должна ожидать сообщения от принтера, что он готов к приему следующего. Это делается двумя способами. При готовности принтер дает импульс в бит подтверждения регистра статуса ввода, адрес порта которого на 1 больше базового адреса принтера. Номер бита подтверждения равен 6 и обычно установлен в 1. Импульс подтверждения сбрасывает этот бит в 0 на достаточно долгое время, чтобы программа на языке ассемблера успела заметить это, если она постоянно следит за регистром.

Другой способ узнать о готовности принтера к приему следующего байта данных состоит в непрерывной проверке бита 7 регистра статуса, который сбрасывается в 0, когда принтер занят, и устанавливается в 1, когда он готов принять данные. Если вы пишете процедуру печати низкого уровня, которая должна работать в интерпретирующем Бейсике или другом очень медленном языке, то применяйте этот метод.

В следующем примере базовый адрес LPT1 берется из области данных BIOS, а затем выводятся данные из буфера, на который указывает регистр BX. Программа постоянно проверяет регистр статуса на занятость и одновременно бит 3 на наличие ошибки на принтере.

```
;---подготовка
          MOV AX,40H          ;ES указывает на область данных BIOS
          MOV ES,AX
          MOV DX,ES:[8]       ;получаем базовый адрес LPT1 в DX
          MOV BX,DATA_ST      ;BX указывает на буфер данных
;---посылаем символ
NEXTC:    MOV AL,[BX]         ;помещаем символ в AL
          OUT DX,AL           ;посылаем символ
          INC  DX             ;DX будет указывать на регистр
          INC  DX             ;управления выводом
          MOV AL,13           ;цепочка битов для импульса строба
          OUT DX,AL           ;посылаем сигнал строба
          DEC AL              ;нормальное состояние регистра
          OUT DX,AL           ;выключаем сигнал строба
;---проверка на ошибку и ожидание готовности принтера
          DEC DX              ;DX указывает на регистр статуса
NOT_YET:  IN   AL,DX          ;получаем байт статуса
          TEST AL,8           ;ошибка?
          JNZ  PR_ERROR       ;переход на обработку ошибки
          TEST AL,80H         ;принтер занят?
```

```
JZ    NOT_YET    ;если занят, то возврат
INC   BX         ;увеличиваем указатель в буфере данных
DEC   DX         ;DX указывает на регистр данных
JMP   NEXTC      ;идем на печать следующего символа
```

Если бит 4 управляющего регистра принтера установлен, значит разрешено прерывание принтера. При работе по прерыванию программа не должна ждать сигнала готовности от принтера, непрерывно опрашивая регистр статуса принтера. Вместо этого она может послать символ и заниматься другими делами; когда принтер будет готов для приема следующего символа, то он пошлет сигнал подтверждения (бит 6 регистра статуса на короткое время будет установлен в 1) и автоматически будет вызвано прерывание принтера. Процедура обработки прерывания пошлет на принтер следующий символ и вернет управление программе, чтобы она могла продолжать свою работу, пока не произойдет следующего прерывания. Когда все данные будут выведены, прерывание должно отключить себя. Прерывание принтера во многом аналогично коммуникационному прерыванию, которое обсуждается в [7.1.8].

К сожалению, оборудование сделано так, что не стоит полагаться на это свойство для первого адаптера принтера. На одних адаптерах оно работает, а на других - нет. Только в случае последовательной/параллельной карты AT вы можете довериться прерыванию. Вместо него можно использовать прерывание таймера, как объяснено в [2.1.7]. Установите микросхему таймера 8253 так, чтобы принтер успевал вывести символ в промежутке между двумя прерываниями. Затем напишите процедуру обработки прерывания, которая посылает на принтер очередной символ всякий раз, когда происходит прерывание времени суток. Для того чтобы обеспечить надежную синхронизацию, заставьте процедуру проверять бит занятости принтера регистра статуса (бит 7), и если принтер еще занят, процедура не должна посылать символ.

6.3.2. Выравнивание правого поля

Реально выравнивание правого поля представляет собой распределение пробелов, находящихся в конце строки, равномерно по промежуткам между всеми словами. Некоторые принтеры имеют специальный режим, который автоматически осуществляет выравнивание. Такую возможность имеет цветной принтер фирмы IBM; посылка управляющего кода 27,77,0 заставляет электронику принтера интерпретировать поступающие данные и форматировать их. В других случаях, когда выводится символ пробела, принтер дол-

жен менять ширину пробелов между словами, переключаясь в графический режим. В графических режимах ширина пробела может изменяться на величину меньше или равную 1/6 от размера символа. К сожалению, многие принтеры на некоторое время останавливаются при переключении между текстовым и графическим режимами, поэтому такой метод может оказаться слишком медленным. Другой способ состоит во вставке обычных символов пробела и распределении их как можно более равномерно по строке. Более сложный графический подход описан ниже.

Шаги, которые необходимо предпринять для форматирования с выравниванием правого поля, могут быть следующими. Во-первых, из установленного формата страницы должно быть вычислено количество символов в строке. Затем необходимо подсчитать число символов, занимаемое каждым из последовательно введенных слов, включая пробелы между словами. Отдельный счетчик должен подсчитывать число пробелов. Когда общая сумма символов превзойдет 80 (или ту ширину принтера, которая установлена), то последнее слово должно быть отброшено из этой суммы вместе с предшествующим ему пробелом. Число оставшихся свободными позиций в строке умножается на 6, поскольку каждый символ занимает размер шести точек по горизонтали, а получившееся число делится на число пробелов между словами.

После печати каждого слова принтер устанавливается в графический режим 480 точек в строке и на него посылается ряд кодов ASCII 0. Каждый такой байт сдвигает печатающую головку на одну точку вправо. Посылаемое число должно быть равно шести для обычного пробела плюс результат распределения пустого пространства. Наконец, если остаток от деления не нулевой, то надо добавить по одному добавочному байту к пробелам, начиная с первого, до тех пор пока остаток не будет исчерпан.

Для примера рассмотрим случай, когда строка состоит из 12 слов, содержащих 61 букву плюс 11 пробелов между словами. Это оставляет в строке с 80 символами 8 свободных позиций. Эти восемь позиций, умноженные на 6 точек, составляют 48 точек дополнительного пространства строки. Поскольку в строке 11 пробелов, то к каждому из них должно быть добавлено по 4 дополнительные точки, после чего останутся еще 4 лишние точки, которые надо добавить по одной к первым четырем пробелам. Тогда первые 4 пробела будут иметь размер 6 точек нормального пробела плюс добавочные 5, что в сумме равно 11. Остальные пробелы этой строки будут размером в 10 точек. Чтобы вывести эти данные на принтер, подготовьте сначала код, посылающий на принтер 1 байт ASCII 0, а затем поместите его в цикл, который должен быть выполнен столько раз, сколько таких байтов нужно вывести. На рис. 6.2 показан этот процесс.

В приведенном ниже примере показаны основы выравнивания по правому полю. Не забудьте об обработке специальных случаев, когда слово длиннее строки (например, длинный ряд тире). Процедура также будет нуждаться в изменении, если строка содержит всего несколько слов, как это бывает в конце параграфа. Не позволяйте ей распределить эти слова равномерно по всей ширине страницы.

Рис. 6.2. Выравнивание по правому краю

Высокий уровень

В следующем примере SPTR указывает на место в буфере данных, с которого начинается следующая строка, выводимая на печать:

```
100 S$ = "This text will be printed with right justification
using the printer alternately in text modes and graphics modes."
110 SPTR = 1                    'указатель в строке данных S$
120 COLS = 1                    'счетчик позиции в строке
130 SPA = 0                     'счетчик пробелов в строке
140 '''вычисляем, сколько слов помещается в строке
150 C$ = MID$(S$,SPTR,1)        'получаем символ
160 IF C$ <> " " THEN 190       'если нет пробела, то двигаемся вперед
170 LASTSPA = COLS              'иначе - запоминаем позицию
180 SPA = SPA + 1               'увеличиваем число пробелов
190 COLS = COLS + 1             'увеличиваем указатель столбца
```

```
200 SPTR = SPTR + 1              'увеличиваем указатель данных
210 IF COLS = 81 THEN 230        'если конец строки, то уходим
220 GOTO 150                     'иначе - переходим к следующему символу
230 IF C$ <> " " THEN 270        'если последний символ не пробел, то уходим
240 COLS = 79                    'иначе - длина строки равна 79
250 SPA = SPA - 1                'отнимаем последний пробел
260 GOTO 340                     'переходим к вычислению пробелов
270 C$ = MID$(S$,SPTR + 1,1)     'проверка на конец слова
280 IF C$ <> " " THEN 300 'если следующий символ не пробел, то уходим
290 GOTO 340                     'иначе - переходим к вычислению пробелов
300 COLS = COLS - LASTSPA        'возвращаемся к концу слова
310 SPTR = SPTR - COLS + 1       'возвращаем указатель
320 COLS = LASTSPA - 1           'убираем последний пробел
330 SPA = SPA - 1                'уменьшаем число пробелов
340 '''вычисляем число точек на пробел
350 EXSPAS = 80 - COLS           'вычисляем число пробелов в конце
360 TOTSPAS = EXSPAS + SPA       'добавляем к остальным пробелам
370 TOTDOT = 6*TOTSPAS           'вычисляем число точек
380 DOTPSP = TOTDOT/SPA          'получаем число точек на пробел
390 EXTDOT = TOTDOT MOD SPA      'остаток от деления
400 '''теперь печатаем первую строчку нашего текста
410 OPEN "LPT1:" AS #1           'открываем принтер
420 PRPTR = 1                    'указатель на начало буфера данных
430 C$ = MID$(S$,PRPTR,1)        'берем символ
440 PRPTR = PRPTR + 1            'увеличиваем указатель
450 IF C$ = " " THEN 500         'если пробел, то на обработку пробела
460 PRINT #1, C$                 'иначе - печатаем символ
470 IF PRPTR = COLS + 1 THEN 590 'выход по концу строки
480 GOTO 430                     'иначе - печатаем следующий символ
490 '''вот процедура печати пробелов
500 PRINT #1, CHR$(27) + "K";    'переход в графический режим
510 NUMDOTS = DOTPSP             'вычисляем число байтов ASCII 0
520 IF EXTDOT = 0 THEN 550       'если нет добавочных точек, уход
530 NUMDOTS = DOTPSP + 1         'иначе - добавляем точку
540 EXTDOT = EXTDOT - 1          'уменьшаем число добавочных точек
550 PRINT #1, CHR$(NUMDOTS);     'посылаем число графических
                                 'символов
560 PRINT #1, CHR$(0);
570 FOR N = 1 TO NUMDOTS: PRINT #1, CHR$(0): NEXT
```

```
580 GOTO 430                    'пробел окончен, переходим к следующему
                                'символу
590 PRINT #1, CHR$(13)          'в конце посылаем символ возврата каретки
```

Низкий уровень

Соответствующая ассемблерная процедура слишком длинная и поэтому здесь не приводится. Она работает точно так же, как и процедура, написанная на Бейсике, за исключением того, что нет необходимости заводить отдельную переменную для хранения строки. Нужно просто установить указатели на начало и конец строки, которую надо напечатать.

6.3.3. Пропорциональная печать

Вообще говоря, для пропорциональной печати требуется специальный принтер, который хранит в ПЗУ информацию о ширине каждого символа. Цветной принтер IBM имеет режим пропорциональной печати, который включается последовательностью 27,78,1, а выключается - 27,78,0. Программа, которая форматирует вывод на такой принтер, должна знать информацию о ширине каждого символа (ее можно найти в документации). Имея эту информацию, она может вычислить, сколько слов поместится на одной строке.

Имейте в виду, что некоторые матричные принтеры автоматически выводят пропорциональный текст в режиме за два прохода. Если слова в строке разделяются добавочными пробелами в графическом режиме, то принтер может переходить ко второму проходу после печати каждого слова, вместо того чтобы повторить сразу всю строку. Поскольку принтеры относительно медленно меняют направление перемещения печатающей головки, то печать текста, выравненного по правому краю, в пропорциональном режиме может занимать очень много времени и оказаться "непосильной ношей" для принтера. Эта проблема не возникает при однонаправленной пропорциональной печати. Отметим, что цветной принтер фирмы IBM может комбинировать пропорциональную печать с автоматическим выравниванием правого края, что делает специальное программирование ненужным.

Опытные программисты могут заставить *любой* графический принтер печатать в пропорциональном режиме. Программа должна иметь в памяти побитовое описание каждого символа (см. [6.3.4]). Вместо того чтобы посылать на принтер код ASCII, который вызывает изображение символа из ПЗУ, используется данная цепочка битов для создания графического изображения строки текста.

Затем вся нужная строка данных выводится на принтер в графическом режиме. Этот подход расходует много памяти на хранение графических образов символов, однако он позволяет полностью контролировать выводимое изображение.

Высокий уровень

В данном примере включается режим пропорциональной печати, а затем выводится первая строка выходных данных программы. Ширина пропорционального шрифта считывается в массив FONTWIDTH из последовательного файла.

```
100 '''считываем массив ширин шрифта
110 DIM FONTWIDTH(127)                    'отводим массив для ширин
120 OPEN "FONTS" FOR INPUT AS #1 'открываем файл ширин, где
130 FOR N = 32 TO 127                     'хранятся ширины для кодов 32-127
140 INPUT #1, FONTWIDTH(N)                'читаем ширину из массива
150 NEXT                                  'следующий элемент
160 '''вычисляем ,сколько символов поместится в строке
170 CHARPTR = 0                           'указатель в буфере
180 LINE$ = ""                            'хранит строку для вывода
190 LINELENGTH = 0                        'счетчик длины в точках
200 WHILE LINELENGTH <480                 'добавляем до заполнения строки
210 C$ = PEEK(BUFPTR + CHARPTR) 'берем символ из буфера данных
220 LINELENGTH = LINELENGTH + FONTWIDTH(ASC(C$))
230 LINE$ = LINE$ + C$                    'добавляем к строке вывода
240 CHARPTR = CHARPTR + 1                 'увеличиваем указатель
250 WEND                                  'переходим к обработке
                                          'следующего символа
260 '''по окончании строки возвращаемся к концу последнего слова
270 IF C$ = "" THEN 310                   'если последний символ пробел, то уход
280 FOR N = LEN(LINE$) TO 1 STEP -1       'идем назад от конца строки
290 IF MID$(LINE$,N,1) = " " THEN 310     'этот символ пробел?
300 NEXT'если нет, то берем следующий
310 LINELENGTH = N - 1                    'если да, то предыдущий - последний
320 '''инициализируем пропорциональную печать и посылаем данные
330 LPRINT CHR$(27);CHR$(78);CHR$(1);     'управляющие коды
340 FOR N = 1 TO LINELENGTH               'для каждого символа
350 LPRINT PEEK(BUFPTR + N-1);            'печатаем его
360 NEXT                                  'переходим к следующему символу
```

Низкий уровень

Программа, написанная на языке ассемблера, должна работать совершенно аналогично программе в приведенном выше примере. Одно из преимуществ ассемблера состоит в том, что для просмотра ширин символов можно использовать инструкцию XLAT. Поместите символ в AL, DS:DX должны указывать на таблицу, после чего можно применять XLAT. Ширина символа будет возвращена в AL.

```
;---просмотр ширин символов
LEA   SI,DATA_BUFFER  ;указываем на буфер данных
LEA   BX,WIDTH_TABLE  ;указываем на таблицу ширин
MOV   AL,[SI]         ;получаем байт данных
XLAT  WIDTH_TABLE     ;получаем его ширину в AL
```

6.3.4. Печать специальных символов

Большинство принтеров не поддерживает расширенный набор символов IBM, однако многие программы используют специальные символы псевдографики. Очень выгодно иметь возможность печатать эти символы, и это не так сложно сделать на любом матричном принтере, который имеет графические возможности. Вместо того чтобы обращаться к ПЗУ принтера, программа должна иметь всю информацию об этих символах и управлять принтером для вывода их на печать.

Сама по себе печать специальных символов тривиальна. Просто разбейте символ на шесть байтов, цепочка битов которых соответствует структуре точек в каждом из шести столбцов, составляющих символ. Например, чтобы напечатать символ горизонтальной двойной черты, код ASCII которого 205, программа должна вывести цепочку битов 00100100 шесть раз в режиме 480 точек в строке. Это количество соответствует ширине символа, поскольку 6/480 равно 1/80 строки. Чтобы перевести принтер именно в этот графический режим, необходимо подать управляющий код 27,75. Затем пошлите число идущих вслед графических данных, которое передается в виде пары байтов, причем младший байт - первый. Наконец, следуют сами 6 бит данных, которые в данном случае равны сумме значений битов 2 и 5 (4 + 32 = 36). Вся последовательность целиком выглядит так: 27, 75, 6, 0, 36, 36, 36, 36, 36, 36. Для более высокого разрешения могут применяться более точные графические режимы; вообще говоря, дополнительные расходы времени машины ничтожны, по сравнению со скоростью операций принтера.

Имеется частная проблема, когда символы псевдографики должны соприкасаться друг с другом по вертикали. Обычно принтеры печатают строку, состоящую из столбца с восемью точками, затем спускаются вниз на высоту 12 точек, оставляя тем самым поле размером в 4 точки между строками символов. Символы псевдографики должны печататься и в этом поле, а в некоторых случаях они занимают в высоту 12 точек. Поскольку большинство печатающих головок имеет только 8 иголок, то единственным решением проблемы является печать таких символов за два прохода, продвигая бумагу перед вторым проходом. В этом случае символ перевода строки (ASCII 10) вообще не используется. Вместо этого принтер попеременно делает интервалы высотой то в 8, то в 4 точки. При втором проходе часть иголок будет на том месте, где уже имеются отпечатанные точки, поэтому надо, чтобы биты для этих иголок были сброшены в 0 (не работали).

Чтобы продвинуть бумагу на высоту четырех точек, надо послать код 27, 65, 4, 27, 50, а на высоту восьми точек - 27, 65, 8, 27, 50. При этом вызывается автоматический возврат каретки. Во время выполнения первого прохода готовится временная строка текста, которая будет печататься при втором проходе. Если данный символ обычный, то в соответствующую позицию временной второй строки символов надо поместить пробел (ASCII 32). Но если встречается специальный графический символ, который должен печататься в четырехточечном поле, то надо поместить его код ASCII в соответствующую позицию второй строки. Например:

Позиция символа	1	2	3	4	5	6	7	8	9	10
Код ASCII	205	32	98	111	114	105	110	103	32	205
Код 2-й строки	205	32	32	32	32	32	32	32	32	205

В памяти должна храниться отдельная таблица цепочек битов этих символов для второго прохода. Для двойной вертикальной черты содержимое таблицы для первого прохода будет 0, 255, 0, 255, 0, 0, а для второго - 0, 15, 0, 15, 0, 0. Отметим, что во втором и четвертом байтах для второго прохода верхние 4 бита сброшены, чтобы не было надпечатки.

Более кратко процесс описывается так, когда начинается печать, то в первую очередь проверяется, является ли данный символ пседографическим, и если нет - он посылается на печать как обычный код ASCII. Во временную строку, используемую для печати второго прохода, вставляется пробел. Затем обрабатывается следующий символ. Когда встречается символ псевдографики, то 6

кодирующих его байтов берутся из таблицы, принтер переводится в графический режим для вывода 6 байт и посылаются данные. Затем принтер автоматически возвращается в текстовый режим. В соответствующую позицию строки для второго прохода помещается код ASCII этого символа псевдографики. Процесс продолжается до конца строки, после чего делается прогон бумаги на высоту четырех точек. При втором проходе надо опять поочередно рассмотреть каждый символ. Если это пробел, то надо печатать символ пробела (т.е. не печатать ничего, а просто продвинуть головку к следующему символу). Если же это графический символ, то надо найти соответствующие ему данные для второго прохода в отдельной таблице и напечатать его таким же образом, как и при первом проходе. Используйте строку для второго прохода для каждой печатаемой строки. На рис. 6.3 показана эта процедура.

Высокий уровень

В следующем примере текст разделен на две колонки, при этом непрерывная линия разделяет страницу посредине. Для простоты здесь печатается только одна строка, однако можно было напечатать и целую страницу, если вставить цикл FOR/NEXT в строках 325 и 505. Для демонстрации двух подходов при первом проходе печатается по одному символу, в то время как при втором проходе печатается целая строка.

```
100 '''таблица данных для первого прохода (только коды 178 и 179)
110 DATA 0, 0, 255, 0, 0, 0
120 DATA 4, 4, 255, 0, 0, 0
130 '''аналогичная таблица для второго прохода
140 DATA 0, 0, 15, 0, 0, 0
150 DATA 0, 0, 15, 0, 0, 0
160 '''помещаем первую таблицу в массив
170 DIM FIRPAS$(45)              'описываем массив данных первого прохода
180 FOR N = 1 TO 2               'заполняем его
190 Y$ = ""                      'Y$ хранит 6 байт на символ
200 FOR M = 1 TO 6: READ X: Y$ = Y$ + CHR$(X): NEXT
210 FIRPAS$(N) = Y$: NEXT        'помещаем в массив
220 '''помещаем в массив вторую таблицу
230 DIM SECPAS$(45)              'описываем массив данных второго прохода
240 FOR N = 1 TO 2               'заполняем массив данными
250 Y$ = ""
260 FOR M = 1 TO 6: READ X: Y$ = Y$ + CHR$(X): NEXT
```

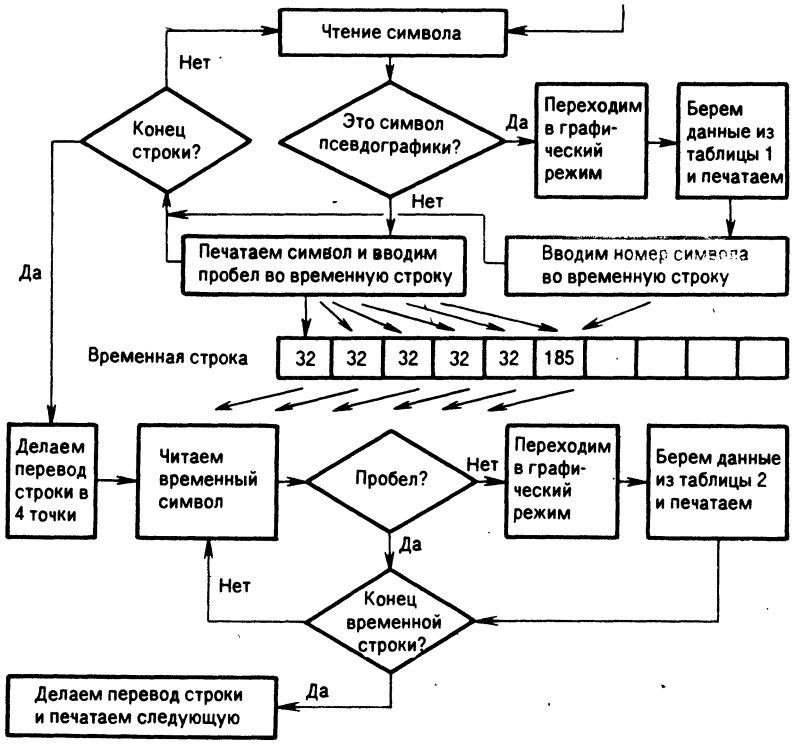


Рис. 6.3. Печать символов псевдографики

```
270 SECPAS$(N) = Y$: NEXT
280 '''печатаем текст следующей строки
290 TEXT$ = "Here is one column" + CHR$(179) + "Here is the second column"
300 TEMP$ = STRING$(80,32)  'создаем строку для 2-го прохода
310 GRAPH$ = CHR$(27) + CHR$(75) + CHR$(6) + CHR$(0)
320 OPEN "LPT1:" AS #1  'открываем принтер
330 FOR N = 1 TO LEN(TEXT$) 'для каждого символа текста
340 C$ = MID$(TEXT$,N,1)  'берем символ и проверяем его
350 IF C$ < CHR$(128) THEN PRINT #1,C$;: GOTO 400
360 '''предполагаем, что все остальные символы - псевдографика
370 PRINT #1,GRAPH$;  'входим в графический режим
380 PRINT #1,FIRPAS$(ASC(C$) - 178) 'выводим 1-й проход
```

```
390 MID$(TEMP$,N) = C$        'маркер в строке 2-го прохода
400 NEXT
410 'смещаемся на 8 точек вниз и делаем 2-й проход
420 PRINT #1,CHR$(27) + CHR$(65) + CHR$(4) + CHR$(141);
430 Z$ = ""                   'Z$ содержит строку для 2-го прохода
440 FOR N = 1 TO LEN(TEXT$)   'для каждого символа текста
450 C$ = MID$(TEMP$,N,1)      'берем символ и обрабатываем его
460 IF C$ = CHR$(32) THEN Z$ = Z$ + " ": GOTO 480
470 Z$ = Z$ + GRAPH$ + SECPAS$(ASC(C$) - 178)
480 NEXT                      'переходим к следующему символу
490 PRINT #1,Z$               'печатаем всю строку сразу
500 PRINT #1,CHR$(10);        'добавляем в конце перевод строки
```

Низкий уровень

Программа на ассемблере использует тот же алгоритм, что и приведенная программа на Бейсике. Если имеется только несколько символов, вы можете сэкономить место, сжав таблицу, с тем чтобы их положение в таблице не было пропорционально их позиции в наборе ASCII. Затем подготовьте небольшую таблицу индексов с помощью инструкции XLAT, которая поможет быстро искать данные в этой таблице.

6.3.5. Копирование содержимого экрана на принтер (дамп экрана)

Дамп текстового экрана сделать достаточно просто, если все используемые символы содержатся в ПЗУ принтера и ни один из них не выводится со специальными атрибутами, такими, как подчеркивание или негативное изображение. В этом простейшем случае программе нужно лишь установить ширину принтера, равной 80 символам, а затем считывать символы поочередно из видеобуфера, посылая их как непрерывный поток данных на принтер. Если в ПЗУ принтера отсутствуют специальные символы, такие, как символы псевдографики, то программа должна подготовить свою таблицу данных для этих символов и выводить их на принтер в графическом режиме. Поскольку эти символы могут заходить в межстрочные интервалы, то может потребоваться специальное программирование (см. [6.3.4]).

Каждый из специальных атрибутов символов создает свои проблемы. Проверяйте атрибут каждого символа при считывании его из видеобуфера (в [4.1.3] обсуждается значение битов, соот-

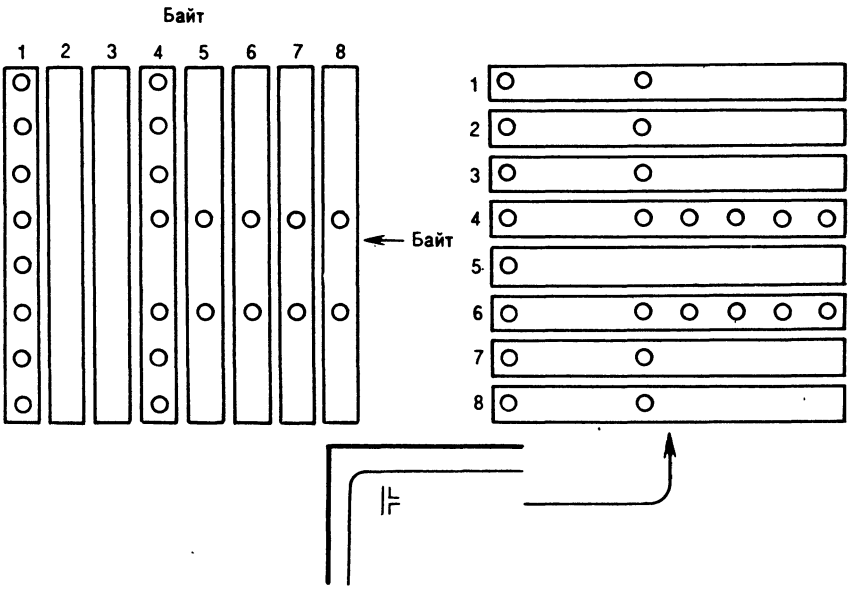


Рис. 6.4. Дамп графического экрана ячейки для одного символа

ветствующее различным атрибутам). Когда символ выделен с помощью подчеркивания или повышенной интенсивности, то надо включать подчеркивание или печать жирным шрифтом на принтере. Однако если символ выводится в негативном изображении, то возникают те же проблемы, что и с некоторыми графическими символами: область негативного изображения должна простираться до верхнего края следующей строки. В этом случае надо, следуя указаниям из [6.3.4], заполнить черным всю область при втором проходе. В зависимости от принтера вам может понадобиться создать специальную таблицу данных для вывода символов в негативе, поскольку, когда они будут печататься, окружающие точки могут находиться слишком близко одна к другой, затемняя изображаемый символ. В этом случае не может быть и речи о печати в два прохода. Простым решением проблемы с негативным изображением является использование графического режима экрана для вывода текста, а затем перевод принтера в один из графических режимов и посылка на него копии графического экрана.

Графические дампы создают свои проблемы. Байт данных принтера соответствует восьми вертикальным точкам, в то время

как на экране байт представляет 8 горизонтальных точек. Поэтому требуется процедура преобразования, показанная на рис. 6.4. Надо сразу получать по 8 байт памяти экрана, выбирая такие, которые соответствуют области точек 8*8. Затем надо использовать логические операции для перестановки битов, как показано в примерах.

Имейте в виду, что большинство матричных принтеров искажают экранное изображение. Это происходит потому, что они работают с масштабным коэффициентом 1:1, в то время как коэффициент экрана 5:6 (масштабный коэффициент сравнивает число горизонтальных точек на дюйм с числом вертикальных точек на дюйм). Точнее говоря, искажение изображения на самом деле возникает из-за масштабного коэффициента экрана, поскольку программы должны специально менять данные для изображения, чтобы оно выглядело так, как нам хочется (например, изображение окружности на экране создается выводом на него эллипса). Когда данные с экрана выводятся на принтер, то эти искажения должны обращаться. Некоторые графические принтеры имеют специальные режимы, в которых можно выводить копию экрана без искажения, а цветной принтер фирмы IBM может менять масштабный коэффициент в любом из своих графических режимов.

Высокий уровень

Приводимая процедура на Бейсике копирует текстовый экран, игнорируя специальные атрибуты:

```
10 OPEN "LPT1:" AS #1                'открываем принтер
20 DEF SEG = &HB000                  'указываем на видеобуфер
30 PRINT #1,CHR$(13)                 'сдвигаем головку влево
40 FOR G = 0 TO 3998 STEP 2          'для каждого байта буфера
50 PRINT #1,CHR$(PEEK(G));           'читаем его и выводим на принтер
60 NEXT                              'обрабатываем следующий байт
```

Переброска цепочек битов для графического дампа в Бейсике требует слишком много времени. Поместите в массив (здесь BYTE$) восемь байтов, отвечающих области экрана 8*8 точек. Создайте второй массив (VERTICAL$) и обнулите его элементы, а затем поочередно перебрасывайте биты элементов этих массивов следующим образом:

```
500 FOR M = 0 TO 7                   'для каждого бита
510 FOR N = 0 TO 7                   'для каждого байта
520 X = ASC(BYTES(N))                'получаем значение байта
```

```
530 Y = 2*(7 - M)                    'маска для одного включенного бита
540 Z = X AND Y                      'проверка этого бита в байте
550 IF Z <> 0 THEN VERTICAL$(M) = CHR$(ASC(VERTICAL$(M) OR 2*N)
                                     'если он включен, то устанавливаем бит
                                     'в соответствующей позиции 2-го массива
560 NEXT N                           'следующий бит
570 NEXT M                           'следующий байт
```

Низкий уровень

Язык ассемблера совершает битовые преобразования намного быстрее. Есть процедура, которая делает эти преобразования очень быстро, поскольку она держит все в микропроцессоре (она немного великовата, но вы можете использовать взамен алгоритм, показанный в Бейсике). Во время работы процедуры 8 результирующих байтов хранятся в регистрах CX, DX, BP и DI. Байт экранных данных помещается в AL, а затем в AH передвигаются последовательно CL, CH, DL и DH. Каждый раз из AL в AH сдвигается один бит и, когда сделаны 4 сдвига, CX и DX обмениваются с DX и BP, после чего все это повторяется снова. Этот процесс повторяется для каждого из восьми экранных байтов и, когда он завершен, преобразованное изображение хранится в регистрах микропроцессора, причем самый левый байт данных для печати - в CL. Содержимое регистров выводится на принтер и обнуляется, после чего процесс повторяется для следующих восьми байтов экрана. Сначала получите 8 байт из видеобуфера и поместите их в буфер с именем BUFFER. Поместите 0 в AX, CX, DX, BP и DI. Затем необходимы следующие действия:

```
        LEA   BX,BUFER   ;указываем на буфер видеоданных
        MOV   SI,0       ;смещение в этом буфере
GETBY:  MOV   AL,[BX][SI] ;берем байт
DOHAF:  XCNG  AH,CL      ;получаем CL, CH, DL и DH
        SHL   AX,1       ;сдвигая бит из AL
        XCNG  AH,CL
        XCNG  AH,CH
        SHL   AX,1
        XCNG  AH,CH
        XCNG  AH,DL
        SHL   AX,1
        XCNG  AH,DL
        XCNG  AH,DH
```

```
        SHL   AX,1
        XCNG AH,DH
;---начинаем вторую половину перемещения битов
        XCNG CX,BP          ;обмениваем содержимое CX и DX
        XCNG DX,DI
        CMP   SI,7          ;если все байты преобразованы, то печатаем
        JE    PRTBS
        INC   SI            ;иначе - переходим к следующему байту
        JMP   SHORT GETBY
;---печатаем байты
PRTBS: PUSH DX              ;сохраняем DX
        MOV   AH,5          ;функция вывода на принтер
        MOV   DL,27         ;код Esc
        INT   21H           ;посылаем его
        MOV   DL,75         ;код графического режима
        INT   21H           ;посылаем его
        MOV   DL,6          ;будет послано 6 байт
        INT   21H
        MOV   DL,0
        INT   21H
        CALL  PR2BS         ;посылаем содержимое CX
        POP   CX
        CALL  PR2BS         ;посылаем содержимое DX
        MOV   CX,BP
        CALL  PR2BS         ;посылаем содержимое BP
        MOV   DX,DI
        CALL  PR2BS         ;посылаем содержимое DI

(переходим к следующей группе из восьми байтов)

PR2BS: PROC NEAR
        MOV   AH,5          ;функция печати
        MOV   DL,CL         ;сначала CL
        INT   21H           ;печатаем
        MOV   DL,CH         ;затем CH
        INT   21H           ;печатаем
        RET
PR2BS  ENDP
```

Глава 7. Ввод/вывод

7.1. Доступ к последовательному порту

При асинхронной связи машина посылает или принимает байты информации порциями по одному биту. Временные интервалы между байтами при этом несущественны, но очень важны интервалы между отдельными битами байта. Сигнал на линии может быть высокого или низкого уровня, что соответствует логическим нулю и единице, и говорят, что линия отмечена (marking), когда уровень высокий, и пустая (spacing), когда уровень низкий.

Линия поддерживается в отмеченном состоянии, когда по ней нет передачи данных. При начале передачи байта данных сигнал падает в 0, отмечая стартовый бит. Затем следуют восемь битов данных (иногда меньше) в виде набора высоких и низких уровней. Последний бит данных может сопровождаться битом четности, используемым для обнаружения ошибок, а затем в последовательность включаются 1 или более стоп-битов, которым соответствует высокий уровень. Эти стоп-биты начинают отмеченное состояние, которое будет сохраняться до тех пор, пока не начнется передача следующего байта данных; число стоп-битов существенно, поскольку они устанавливают минимальное время, которое должно пройти перед следующим стартовым битом. На рис. 7.1 показана эта последовательность.

Конечно, передающая и приемная станции должны применять один и тот же протокол для цепочек битов и работать с одной и той же скоростью обмена (измеряемой в битах в секунду, называ-

емых также бодами). При обмене могут легко возникать ошибки, поэтому коммуникационное оборудование предоставляет разнообразную информацию о статусе как самого порта, так и присоединенного к нему модема. Задачей модема является преобразование сигнала, генерируемого портом коммуникации, в акустический сигнал, который может затем быть передан по телефонному каналу. Большинство модемов предоставляет также дополнительные коммуникационные возможности, такие, как автоматический вызов и ответ, которые не поддерживаются самим портом коммуникации.

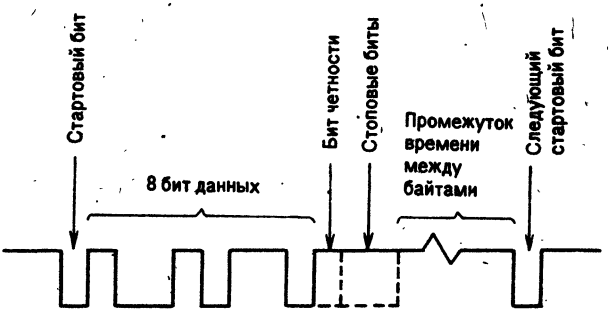


Рис. 7.1. Передача одного байта последовательных данных

7.1.1. Программирование микросхемы UART 8250

Последовательная связь настолько сложна, что были разработаны специальные микросхемы, выполняющие работу по формированию и синхронизации строк битов, составляющих последова-

тельные данные. Такие микросхемы называют *универсальным асинхронным приемником-передатчиком* (universal asynchronous receiver transmitter или UART). IBM PC использует UART 8250 фирмы Intel.

Операционная система поддерживает 2 порта коммуникации, поэтому в машине имеются 2 микросхемы. Их базовые адреса хранятся в ячейках 0040:0000 для COM1 и 0040:0002 для COM2. (Базовый адрес - это двухбайтовый адрес порта, который является младшим из группы адресов портов, дающих доступ к UART.) На всех машинах, кроме PCjr, COM1 имеет базовый адрес 3F8H, а COM2 - 2F8H; PCjr имеет свой внутренний модем по адресу 3F8H, а COM1 - по адресу 2F8H. Для удобства мы в дальнейшем будем всегда нумеровать регистры 3FxH, но все сказанное в равной степени применимо и к регистрам 2FxH.

Микросхема 8250 имеет 10 программируемых однобайтовых регистров, с помощью которых управляется и контролируется порт коммуникации. Большинство из них занимается инициализацией порта, процессом, в котором возникает много сложностей. Доступ к этим 10 регистрам осуществляется через семь адресов портов с номерами 3F8H - 3FEH (или 2F8H - 2FEH). В пяти случаях регистр, к которому получаем доступ через данный порт, зависит от того, как установлен бит 7 в регистре контроля линии, который является единственным регистром с адресом порта 3FBH. Вот эти регистры:

3F8H (OUT, бит 7 = 0 в 3FBH)	Регистр хранения передатчика
3F8H (IN, бит 7 = 0 в 3FBH)	Регистр данных приемника
3F8H (OUT, бит 7 = 1 в 3FBH)	Делитель скорости обмена (младший)
3F9H (IN, бит 7 = 1 в 3FBH)	Делитель скорости обмена (старший)
3F9H (OUT, бит 7 = 0 в 3FBH)	Регистр разрешения прерывания
3FAH (IN)	Регистр идентификации прерывания
3FBH (OUT)	Регистр управления линии
3FCH (OUT)	Регистр управления модемом
3FDH (IN)	Регистр статуса линии
3FEH (IN)	Регистр статуса модема

Из десяти регистров только шесть необходимы для простой последовательной связи. Регистр хранения передатчика содержит байт данных, которые будут посланы [7.1.6], а регистр данных приемника - последний полученный байт данных [7.1.7]. Регистры управления и статуса линии инициализируют и управляют линией связи, используя скорость обмена, содержащуюся в двух регистрах делителя скорости обмена [7.1.2]. Из оставшихся четырех регис-

тров регистры управления и статуса модема необходимы только для связи через модем [7.1.5], а два регистра, связанных с прерываниями - только в процедурах, управляемых прерываниями [7.1.8].

Прерывания при связи используют в целях эффективности. Обычная коммуникационная процедура непрерывно проверяет регистр статуса линии, ожидая вводимого символа или указания, что все готово для передачи следующего байта данных. Поскольку процессор работает очень быстро, по сравнению с обычными скоростями, с которыми передаются последовательные данные, то этот метод напрасно расходует процессорное время, которое может использоваться для обработки поступающих или передаваемых данных. По этой причине микросхема 8250 может быть установлена в режим, вызывающий прерывание при появлении символа, возникновении ошибки и т.п. Это прерывание моментально вызовет процедуру вашей программы, которая, скажем, будет передавать следующий символ из коммуникационного буфера.

7.1.2. Инициализация последовательного порта

При инициализации ("открытии") порта коммуникации устанавливаются все его параметры. Эти параметры включают в себя длину слова, число стоп-битов, установку четности и скорость обмена. Длина слова - это число битов, которое образует основную единицу данных. Хотя привычно работать с порциями по 8 бит, но для стандартных файлов ASCII (в которых все символы имеют коды, не превышающие ASCII 128) достаточно 7 бит, а для передачи численных данных - всего 4 бит.

Высокий уровень

Бейсик открывает коммуникационный канал как файл, и поэтому ему должен быть присвоен идентификационный номер:

```
OPEN "COM1: ....." AS #1
```

В кавычках должна быть помещена вся информация, необходимая для инициализации порта коммуникации, при этом каждый элемент отделяется от предыдущего запятой. Инициализирующие данные всегда вводятся в следующем порядке:

Скорость обмена	- дается как целое число: 75, 100, 150, 300, 600, 1200, 1800, 2400, 4800 или 9600 бод. По умолчанию берется скорость обмена 300 бод.
Четность	- вводится как односимвольный код: O для нечетной четности, E - для четной и N - при отсутствии контроля по четности. Могут быть также S, когда бит четности всегда равен 0, и M, когда бит четности всегда равен 1. Если используются 8 бит данных, то надо указывать N; при передаче четырех бит не надо использовать N. По умолчанию принимается E.
Биты данных	- дается как целое число 4, 5, 6, 7 или 8. По умолчанию берется 7.
Стоп-биты	- дается как целое число 1 или 2, причем 2 значение по умолчанию для 75 и 110 бод, а 1 - для остальных. Когда число битов данных равно 4 или 5, то 2 обозначает 1 1/2 стоп-бита. Такое значение возможно при коммуникации, так как в этом случае бит является единицей времени и поэтому делим.

Оператор OPEN "COM1:" AS #1 открывает COM1 для связи со скоростью 300 бод с четной четностью, передавая 7 бит данных и 1 стоп-бит. OPEN "COM1:1200,O,8,1" устанавливает скорость 1200 бод, нечетную четность, 8 бит на символ и 1 стоп-бит. Отметим, что вы можете завершить оператор OPEN выражением LEN = число, где число устанавливает максимальный размер блока, с которым могут работать операторы GET и PUT (по умолчанию 128 байт). Имеется ряд команд управления модемом, которые также могут быть включены в эту спецификацию (в [7.1.5] объясняется специальная терминология, применяемая здесь).

RS	Подавляет сигнал "Запрос на посылку" (Request to send).Если эта команда опущена, то OPEN "COM" включает RTS.
CS	Вызывает проверку линии "Очистка посылки" (Clear to send). За этой командой может следовать значение (от 0 до 65535), определяющее число миллисекунд, которые проистекут до момента выдачи сигнала ошибки тайм-аута, например CS500. Значение по умолчанию принято 1000, и 0, если указан параметр RS.
DS	Вызывает проверку линии "Готовность набора данных" (Data set ready). Допускается необязательный параметр, как и для CS. Значение по умолчанию - 1000.
CD	Вызывает проверку линии "Определение носителя" (Carrier detect). Допускается необязательный временной параметр, как и для CS. Значение по умолчанию - 0.

LF	Вызывает автоматическую подачу кода перевода строки (ASCII 10) после каждого символа возврата каретки (ASCII 13). Используется для последовательного вывода на принтер.
PE	Разрешает проверку четности, вызывая ошибку тайм-аута устройства при возникновении ошибки четности.

Эти специальные команды могут помещаться в любом месте оператора OPEN "COM" и в любом порядке. Отметим, что для выполнения оператора OPEN сигналы CTS и DSR должны быть установлены, в противном случае будет выдана ошибка тайм-аута устройства. В заключение приводим оператор OPEN "COM", содержащий все параметры, кроме RS и LF:

```
OPEN "COM1:1200,O,7,1,CS2000,DS2000,CD,PE" AS #1 LEN = 256
```

Средний уровень

Функция 0 прерывания 14H BIOS инициализирует порт коммуникации. В DX должен находиться номер коммуникационного канала (COM1 = 0, COM2 = 1). В AL должен содержаться байт инициализирующих данных, значение битов которого может быть следующим:

биты	1-0	длина слова. 10 = 7 бит, 11 = 8 бит
	2	число стоп-битов. 0 = 1, 1 = 2
	4-3	четность. 00 или 10 = нет, 01 = нечетн., 11 = четн.
	7-5	скорость обмена. 000 = 110 бод
		001 = 150 бод
		010 = 300 бод
		011 = 600 бод
		100 = 1200 бод
		101 = 2400 бод
		110 = 4800 бод
		111 = 9600 бод

В данном примере порт инициализируется со словом, равным 8 битам, одним стоп-битом и четной четностью. Скорость обмена - 1200 бод.

```
;---присваиваем значения параметров переменным
MOV  WORDLENGTH,00000011B   ;длина слова 8 бит
MOV  STOPBITS,00000000B     ;1 стоп-бит
MOV  PARITY,00011000B       ;четная четность
MOV  BAUDRATE,10000000B     ;скорость 1200 бод
```

```
;---инициализируем COM1
MOV  AL,0                ;чистим AL
OR   AL,WORDLENGTH       ;устанавливаем нужные биты
OR   AL,STOPBITS
OR   AL,PARITY
OR   AL,BAUDRATE
MOV  AH,0                ;функция инициализации порта
MOV  DX,0                ;выбираем COM1
INT  14H                 ;инициализируем порт
```

Низкий уровень

Независимо от того, занимаемся мы вводом или выводом, как минимум, 4 регистра микросхемы 8250 должны быть инициализированы для операций обмена. Это регистры делителя скорости обмена, регистр контроля линии и регистр разрешения прерывания.

Инициализация скорости обмена. Делитель скорости обмена - это число, на которое надо разделить частоту системных часов (1190000 Гц), чтобы получить желаемую скорость обмена. Например, для скорости обмена 1200 бод делитель скорости обмена должен быть равен 96, поскольку 1190000/96 приближенно равно 1200. Чем больше делитель, тем меньше скорость обмена. Скорость обмена 300 бод и меньше требует двухбайтового числа для делителя. Старший байт посылается в 3F9H (или 2F9H), а младший - в 3F8H (2F8H). В обоих случаях бит 7 регистра управления линии должен быть установлен в 1 перед засылкой значений, в противном случае по этим двум адресам значения будут адресованы в другие регистры (см. [7.1.0]). Вот некоторые значения, требуемые для обычных скоростей обмена:

Скорость обмена	3F9H	3F8H
110	04H	17H
300	01H	80H
600	00H	C0H
1200	00H	60H
1800	00H	40H
2400	00H	30H
3600	00H	20H
4800	00H	18H
9600	00H	0CH

. Всегда устанавливайте регистры скорости обмена первыми, так как они единственные, которые требуют установки бита 7 в регистре контроля линии. После этого надо изменить содержимое регистра контроля линии, сбрасывая бит 7, чтобы все остальные доступы к регистрам были правильными. Поскольку регистр контроля линии является регистром только для записи, то нет способа вернуть бит 7 обратно в 1 без одновременной установки всех остальных битов этого регистра. Отметим, что PCjr использует другие делители, описание которых вы можете найти в техническом руководстве.

Инициализация регистра контроля линии. Значения битов регистра контроля линии, адрес порта которого равен 3FBH (или 2FBH), могут быть следующими:

биты 1-0 длина символа. 00 = 5 бит, 01 = 6 бит, 10 = 7 бит, 11 = 8 бит

2 число стоп-битов. 0 = 1, 1 = 1,5, если длина символов равна пяти, иначе - 2

3 четность. 1 = генерируется бит четности, 0 = нет

4 тип четности. 0 = нечетная, 1 = четная

5 фиксация четности. Заставляет бит четности всегда быть 0 или 1.
0 = отменена
1 = всегда 1, если бит 3 = 1 & бит 4 = 0
или 1 = всегда 0, если бит 3 = 1 & бит 4 = 1
или 1 = нет четности, если бит 3 = 0

6 установка перерыва. Вызывает вывод строки нулей в качестве сигнала отдаленной станции.
0 = запрещено, 1 = перерыв

7 меняет адреса портов других регистров

Обычно биты 5-7 сброшены в 0. Остальные описывают значения, определяемые протоколом обмена.

Регистр разрешения прерывания. Даже если вы не используете прерывания, все равно нужно произвести запись в регистр разрешения прерывания, чтобы быть уверенным, что прерывания запрещены. Просто поместите в этот регистр 0. Регистр идентификации прерывания можно игнорировать.

Инициализация остальных регистров связана с модемами. Ясно, что модемы нужны только для связи с удаленными устройствами, а не для управления близлежащими, такими, как последовательный принтер. В [7.1.5] объяснено, как инициализировать регистр контроля модема.

В следующем примере из области данных BIOS берется базовый адрес COM1, после чего различные регистры инициализиру-

ются для скорости обмена 1200 бод, семибитовых данных, четной четности и одного стоп-бита.

```
;---получаем базовый адрес COM1
MOV  AX,40H          ;ES указывает на область данных BIOS
MOV  ES,AX
MOV  DX,ES:[0]       ;получаем базовый адрес COM1
;---инициализируем регистры делителя скорости обмена на 1200 бод
ADD  DX,3            ;указываем на регистр контроля линии
MOV  AL,10000000B    ;устанавливаем бит 7
OUT  DX,AL           ;посылаем байт
DEC  DX              ;указываем на старший байт делителя
DEC  DX              ;скорости обмена
MOV  AL,0            ;старший байт для 1200 бод
OUT  DX,AL           ;посылаем старший байт для 1200 бод
DEC  DX              ;указываем на младший байт делителя
MOV  AL,60H          ;младший байт делителя для 1200 бод
OUT  DX,AL           ;посылаем младший байт
;---инициализируем регистр контроля линии
MOV  AL,0            ;обнуляем AL
OR   AL,10B          ;длина данных 7 бит
OR   AL,000B         ;1 стоп-бит
OR   AL,1000B        ;генерируется бит четности
OR   AL,10000B       ;четная четность
ADD  DX,3            ;указываем на регистр контроля линии
OUT  DX,AL           ;посылаем инициализационное значение
;---инициализируем регистр разрешения прерывания
DEC  DX              ;указываем на регистр разрешения
DEC  DX              ;прерывания
MOV  AL,0            ;запрещаем прерывания
OUT  DX,AL           ;посылаем байт
```

7.1.3. Установка текущего коммуникационного порта

Имеются два способа, с помощью которых программа может определить, какой из коммуникационных портов должен использоваться. Один из них состоит в указании номера канала в операторе программы. Второй способ состоит в написании программы для обмена через порт COM1, но изменении коммуникационного адаптера, доступ к которому идет через COM1.

Область данных BIOS содержит место для четырех двухбайтовых переменных, в которых находятся базовые адреса коммуникационных каналов (MS-DOS поддерживает только первые два из них). Базовый адрес порта - это младший из группы адресов портов, через которые можно получить доступ к данному коммуникационному каналу. Базовый адрес для COM1 хранится в ячейке 0040:0000, а для COM2 - в ячейке 0040:0002. Для смены коммуникационных портов надо просто поменять эти два значения. Повторная смена значений приведет к первоначальному назначению портов.

Высокий уровень

В Бейсике оператор OPEN "COM" может иметь вид: OPEN C$ + "1200,N,8" AS #2, где C$ может быть либо "COM1:", либо "COM2:". В качестве альтернативы также используют PEEK и POKE для обмена базовых адресов:

```
100 DEF SEG = &H40                    'указываем на область данных BIOS
110 X = PEEK(0): Y = PEEK(1)          'запоминаем первые 2 байта
120 POKE 0,PEEK(2): POKE 1,PEEK(3)    'переносим следующие два байта
130 POKE 2,X: POKE 3,Y                'засылаем запомненные значения
```

Средний уровень

Если программа обращается к коммуникационному порту через прерывание 14H BIOS, то COM-порт определяется содержимым DX, которое равно 0 или 1 (для COM1 или COM2). Вместо того чтобы присваивать DX непосредственное значение, заполняйте его из переменной, которой может быть присвоено значение 0 или 1. Программы, использующие коммуникационные функции 3 и 4 прерывания 21H, всегда адресуются к COM1. В этом случае надо поменять базовые адреса:

```
;---обмен базовых адресов для COM1 и COM2
MOV  AX,40H          ;ES указывает на область данных BIOS
MOV  ES,AX
MOV  DX,ES:[0]       ;помещаем 1-й базовый адрес в DX
MOV  AX,ES:[2]       ;помещаем 2-й базовый адрес в AX
MOV  ES:[0],AX       ;обмениваем адреса
MOV  ES:[2],DX
```

7.1.4. Определение статуса коммуникационного порта

Регистр статуса линии микросхемы UART 8250 определяет протокол связи. Этот регистр имеет адрес порта на 5 больше, чем базовый адрес данного канала. Обычно он постоянно просматривается в процессе коммуникационного обмена. При передаче данных регистр сообщает, что предыдущий символ уже послан, позволяя программе записать новый символ поверх его. При приеме данных регистр информирует программу о поступлении следующего символа, с тем чтобы программа могла прочитать его прежде, чем он будет уничтожен следующим прибывшим. Значения битов этого регистра следующие:

бит	0	1 = байт данных получен
	1	1 = полученные данные были перезаписаны (предыдущий символ не был вовремя считан)
	2	1 = ошибка четности (вероятно, из-за шума в линии)
	3	1 = ошибка окружения (передача не синхронизована)
	4	1 = обнаружен перерыв (получена длинная строка единиц, индицирующая, что другая станция запрашивает конец передачи)
	5	1 = регистр хранения передатчика пуст (в этот регистр должны помещаться передаваемые данные)
	6	1 = регистр сдвига передатчика пуст (этот регистр получает данные из регистра хранения и преобразует их в последовательный вид)
	7	1 = тайм-аут (устройство не связано с машиной)

Высокий уровень

В Бейсике сначала определите базовый адрес используемого коммуникационного порта, затем добавьте к нему 5 и примените оператор INP для получения байта из этого порта. В приложении Б объясняется, как в Бейсике производятся битовые операции, которые необходимо проделать программе, чтобы интерпретировать значение этого байта. В следующем примере проверяется бит наличия перерыва:

```
100 DEF SEG = &H40                'указываем на область данных BIOS
110 ADDR = PEEK(4) + PEEK(5)*256  'вычисляем адрес COM2
120 X = INP(ADDR + 5)             'вычисляем адрес регистра статуса
130 IF X AND 16 THEN 500          'переход на подпрограмму, если бит 4 = 1
.
.
500 '''начинаем процедуру обработки перерыва
```

Средний уровень

Функция 3 прерывания 14H BIOS возвращает в AH регистр статуса линии (AL получает регистр статуса модема [7.1.5]). При входе DX должен содержать номер коммуникационного порта, к которому осуществляется доступ, где COM1 = 0, а COM2 = 1. Как и в предыдущем, в этом примере проверяется наличие перерыва:

```
MOV  AH,3              ;номер функции
MOV  DX,1              ;выбираем COM2
INT  14H               ;получаем байт статуса
TEST AH,10000B         ;обнаружен перерыв?
JNZ  BREAK_DETECT      ;если да, то переходим к процедуре обработки
```

Низкий уровень

Следующий пример совершенно аналогичен приведенному на Бейсике. Из области данных BIOS считывается базовый адрес коммуникационного канала, к нему добавляется 5, а затем из полученного адреса порта считывается байт статуса.

```
MOV  AX,40H            ;ES указывает на область данных BIOS
MOV  ES,AX
MOV  DX,ES:[2]         ;получаем базовый адрес COM2
ADD  DX,5              ;добавляем 5 для регистра статуса
IN   AL,DX             ;получаем байт статуса
TEST AL,10000B         ;бит 5 установлен?
JNZ  BREAK_DETECT      ;если да, то переходим к обработке перерыва
```

7.1.5. Инициализация и управление модемом

Имеется 6 линий, по которым модемы связываются с компьютером (усовершенствованные модели могут иметь добавочные линии по интерфейсу RS232). Вот их названия, сокращения и функции:

От компьютера к модему

Data Terminal Ready (DTR) (готовность компьютера)	Информирует модем, что компьютер включен и готов к связи
Request To Send (RTS) (запрос на посылку)	Информирует модем, что компьютер ожидает посылки данных

От модема к компьютеру

Data Set Ready (DSR) (готовность модема)	Информирует компьютер, что модем включен и готов
Clear To Send (CTS) (готовность к посылке)	Информирует компьютер, что модем готов начать передачу данных
Data Carrier Detect (DCD) (обнаружен носитель данных)	Информирует компьютер, что модем связан с другим модемом
Ring Indicator (RI) (индикатор звонка)	Информирует компьютер, что телефонная линия, к которой присоединен модем, имеет звонок

Сначала компьютер устанавливает сигнал DTR, а затем инструктирует модем связаться с удаленной станцией. После того как модем установил связь, он устанавливает сигнал DSR. Этот сигнал информирует компьютер, что модем готов к связи и в этот момент компьютер может установить сигнал RTS. И если модем ответит сигналом CTS, начнется передача.

Две стандартные линии, по которым компьютер управляет модемом, доступны через регистр контроля модема микросхемы UART 8250. Этот регистр имеет адрес порта на 4 больше, чем базовый адрес используемого коммуникационного канала. Значения его битов таковы:

Регистр контроля модема

биты	7-5	(всегда 0)
	4	1 = выход UART замкнут на вход
	3	добавочный пользователь назначен на вывод #2
	2	добавочный пользователь назначен на вывод #1
	1	1 = "запрос на посылку" активен
	0	1 = "готовность компьютера" активна

Обычно биты 0 и 1 регистра контроля модема установлены, а остальные - равны 0. Бит 2 равен 0, за исключением случаев, когда производитель модема предназначил его для специальных нужд. Бит 3 установлен только в случае, когда используются прерывания [7.1.8]. Наконец, бит 4 предоставляет возможность тестирования коммуникационных программ без установления реальной связи. Выходной сигнал микросхемы UART подается на вход, как будто UART принимает последовательные данные. Это свойство можно применять для тестирования правильности работы самой микросхемы. Однако при использовании коммуникационных процедур прерывания 14H BIOS оно недоступно.

Четыре линии, по которым модем посылает информацию компьютеру, управляются регистром статуса модема. Этот регистр расположен по адресу порта на 6 больше, чем базовый адрес используемого коммуникационного адаптера. Вот значение его битов:

Регистр статуса модема

бит	7	1 = DCD
	6	1 = RI
	5	1 = DSR
	4	1 = CTS
	3	1 = изменение в DCD
	2	1 = изменение в RI
	1	1 = изменение в DSR
	0	1 = изменение в CTS

Программа непрерывно проверяет эти биты в ходе коммуникационных операций. Отметим, что 4 младших бита параллельны старшим четырем битам. Эти биты устанавливаются в 1 только тогда, когда происходит изменение в статусе соответствующего старшего бита с момента, когда регистр читался последний раз. Все 4 младших бита автоматически сбрасываются при чтении регистра. Программы любого уровня могут непосредственно читать этот регистр. Другой возможностью является использование функции 3 прерывания 14H BIOS, которая возвращает регистр статуса модема в AL (при этом в AH будет содержаться регистр статуса линии). При входе DX должен содержать номер коммуникационного канала (0 или 1).

Многие модемы имеют гораздо больше возможностей по сравнению с теми, что отражены в двух связанных с модемом регистрах! Имеются возможности автоматической связи и автоматического ответа, которые контролируются управляющей строкой. Эта строка посылается в модем в виде обычных данных. Модем выделяет управляющую строку из данных по специальному символу, используемому только для указания начала управляющей строки. Этот символ может быть предопределенным (часто используется код Esc - ASCII 27) или выбираемым пользователем. Модем способен определить, насколько длинной должна быть каждая строка, поэтому по окончании строки он опять рассматривает входящий поток информации как данные. Каждый модем имеет свой набор команд. В качестве примера рассмотрим команды, используемые внутренним модемом PCjr:

Символ	Значение	Применение
A	ответ	вход в режим ответа
Bn	перерыв	посылает сигнал перерыва n x 100 мс
Cn	отсчет n	отсчитывает n звонков до ответа
Dn...n	вызов	посылает строку чисел n...n
Fn	формат	устанавливает протокол связи
H	разрыв	прекращает связь с машиной
I	инициализация	инициализирует модем
LR	долгий ответ	меняет используемую кодовую систему
M	режим	модем принимает символы как данные
Nn	новый	меняет командный символ на n
O	originate	вход в режим originate
P	pick-up	вход в режим голоса
Q	запрос	запрос статуса модема
R	повтор	повторить команду связи
Sn	скорость	выбор скорости обмена
Tn...n	прозрачность	игнорировать управляющие строки в следующих n...n байтах
V	голос	перевести модем в режим голоса
W	ожидание	ничего не делать до следующей команды
X	передать	передача тона вызова
Z	тест	проводит диагностику оборудования

В ответ на команду запроса модем выдает информацию о своем состоянии, посылая ее в UART как обычные данные. Помимо прочей информации может сообщаться, что линия занята. Чтобы правильно использовать команды управления модемом и информацию о его статусе, надо тщательно изучить документацию на данный модем. Модем PCjr описан в техническом руководстве по PCjr. Приведенные ниже примеры дают только общую схему установления связи через модем.

Высокий уровень

Поскольку телефонная линия очень медленная, то связь с модемом это одна из областей, где программирование на Бейсике ничем не хуже, чем на языке ассемблера. Приведем упрощенную схему:

```
100 OUT BASEADDRESS + 4,1          'устанавливаем бит DTR
110 '''теперь посылаем управляющую строку для вызова и установле-
120 '''ния связи - этот код меняется от модема к модему
```

```
200 X = INP(BASEADDRESS + 2)        'получаем регистр статуса модема
210 IF X AND 2 <> 2 THEN 200        'ждем пока будет установлен бит 1
220 OUT BASEADDRESS + 4,3           'устанавливаем бит RTS
230 X = INP(BASEADDRESS + 2)        'получаем регистр статуса модема
240 IF X AND 1 <> 1 THEN 230        'ждем пока будет установлен бит 0
250 '''теперь посылаем данные
```

Низкий уровень

Схема связи через модем на языке ассемблера выглядит так:

```
;---устанавливаем сигнал DTR
        MOV DX,BASADR    ;начинаем с базового адреса
        ADD DX,4         ;указываем на регистр контроля модема
        MOV AL,1         ;устанавливаем бит 1
        OUT DX,AL        ;посылаем в порт
;---посылаем управляющую строку модему для вызова

(этот код различен для разных типов модемов)

;---ожидаем пока будет установлен сигнал DSR
        INC  DX          ;указываем на регистр статуса модема
        INC  DX
TRYAG:  IN   AL,DX       ;получаем содержимое
        TEST AL,10B      ;проверяем второй бит
        JZ   TRYAG       ;ждем пока он не будет равен 1
;---устанавливаем бит RTS
        DEC DX           ;возвращаемся к регистру управления
        DEC DX
        MOV AL,3         ;устанавливаем сигнал RTS
        OUT DX,AL        ;посылаем в порт
;---ожидаем сигнала CTS
        INC  DX          ;возвращаемся к регистру статуса
        INC  DX
ON_MO:  IN   AL,DX       ;получаем байт статуса
        TEST AL,1        ;проверяем бит CTS
        JZ   ON_MO       ;не продолжаем, пока он не установлен
;---теперь можно посылать данные
```

7.1.6. Передача данных

Передача данных проще, чем их прием, поскольку программа полностью контролирует состав данных и скорость, с которой они должны посылаться. Тем не менее процедуры передачи могут быть достаточно сложными, если они обрабатывают данные по мере того, как они посылаются. Могут быть также проблемы с синхронизацией при использовании протокола XON/XOFF. Этот протокол использует коды ASCII 17(XON) и 19(XOFF), для того чтобы сигнализировать принимающей станции, что передатчик хочет продолжить передачу временно прерванного потока данных. Чтобы принять эти сигналы, программа должна непрерывно анализировать принимаемые символы при передаче (в полнодуплексном режиме, в котором обычно работают модемы, сигналы одновременно идут в обе стороны по телефонному каналу). Кроме того, чтобы обнаружить, что удаленная станция посылает строку нулей в качестве сигнала перерыва, должен непрерывно анализироваться статус бита перерыва (номер 4) регистра статуса линии [7.1.4]. На рис. 7.2 (см. [7.1.7]) показано, как процедура передачи данных взаимодействует с кодом, принимающим данные.

Вследствие этих причин представленные здесь процедуры, отдельно передающие данные, являются искусственными. Но их можно скомбинировать с процедурами приема данных, описанными в [7.1.7], для создания общего представления. Ясно, что для создания действующей процедуры необходимо затратить большие усилия, особенно в части обнаружения и исправления ошибок при передаче данных.

Высокий уровень

Для того чтобы в Бейсике послать данные в открытый коммуникационный порт, надо использовать операторы PRINT#, PRINT# USING или WRITE#. Последние два оператора имеют специальный формат, параллельный применяемому ими при выводе на дисплей. Обычно используется оператор PRINT#. В следующем примере посылаемые данные берутся непосредственно с клавиатуры. Предполагается, что COM1 уже открыт, как показано в [7.1.2]. Процедура обрабатывает бит перерыва в регистре статуса линии.

```
500 C$ = INKEY$: IF C$ <> "" THEN PRINT #1,C$
                                    'если клавиша нажата, посылаем ее
510 X = INP(BASEADDRESS + 5)        'читаем регистр статуса линии
```

```
520 IF X AND 32 = 32 THEN 1000      'проверяем бит перерыва
530 IF EOF(1) THEN 500              'если буфер пуст, то ждем ввода

(здесь расположена процедура приема данных)

1000 '''здесь процедура обработки перерыва
```

Средний уровень

Функция 1 прерывания 14H BIOS посылает символ, находящийся в AL, в коммуникационный канал. При входе DX содержит номер порта (0 или 1). При возврате AH получает байт статуса, в котором бит 7 = 1, если операция неуспешна. В этом случае имеют значение следующие биты:

бит 4	обнаружен перерыв (сигнал "стоп" от принимающей станции)
5	регистр сдвига передатчика пуст
6	регистр хранения передатчика пуст

MS-DOS имеет функцию для передачи по коммуникационному каналу символа, помещаемого в DL. Это функция номер 4 прерывания 21H, но у нее нет никаких преимуществ перед функцией BIOS; она не возвращает статусной информации и не позволяет назначать, какой из коммуникационных портов надо использовать (это всегда COM1).

Чтобы вывести строку данных, воспользуйтесь функцией 40H прерывания 21H. Это обычная функция вывода для всех файлов и устройств методом доступа дескриптора файлов. COM1 имеет предопределенный номер - #3. Поместите номер файла в BX, а число передаваемых байтов в CX. Пусть DS:DX указывают на буфер выводимых данных, затем вызывайте функцию.

```
MOV  AH,40H          ;номер функции
MOV  BX,3            ;предопределенный номер файла для COM1
MOV  CX,50           ;выводим 50 байт
LEA  DX,BUFER        ;DS:DX указывают на буфер данных
INT  21H             ;посылаем данные
JC   COM_ERROR       ;уход на обработку ошибки
```

Отметим, что при использовании предопределенных номеров файлов их не надо открывать. Если произошла ошибка, устанавли-

вается флаг переноса, а в AX возвращается 5, если коммуникационный порт не готов, и 6 при указании неверного номера файла.

Низкий уровень

Когда байт данных помещается в регистр хранения передатчика, то он автоматически выводится в последовательный канал через регистр сдвига передатчика, который сериализует данные. Нет необходимости в импульсе бита строба, как это делается в случае параллельного адаптера. Бит 5 регистра статуса линии показывает, свободен ли регистр хранения передатчика для приема данных. Регистр постоянно проверяется до тех пор, пока бит 5 не станет равным 1. После этого в регистр хранения передатчика посылается очередной байт. В процессе передачи бит 5 равен 0, и только когда он опять станет равным 1, в регистр хранения передатчика может быть послан следующий символ. Этот процесс повторяется до тех пор, пока в этом есть необходимость.

В следующем примере даны основные понятия об этой процедуре. Конечно, она может быть создана необычайно сложной (в частности, программирование связи требует особо тщательных процедур обнаружения ошибок и восстановления при сбоях). В примере предполагается, что коммуникационный порт и модем уже инициализированы, как показано в [7.1.2] и [7.1.5]. Первая часть это цикл проверки ошибок и приема символов. В [7.1.7] приведен код для процедуры приема данных.

```
;---ждем, пока все будет готово для посылки символа
KETRY: MOV   DX,BASADR     ;базовый адрес
       ADD   DX,5          ;указываем на регистр статуса линии
       IN    AL,DX         ;получаем байт статуса
       TEST  AL,00011110B  ;проверяем на ошибку
       JNZ   ERRROUT       ;если есть, то на процедуру обработки ошибки
       TEST  AL,00000001B  ;проверяем, получены ли данные
       JNZ   RECEIVE       ;если да, то на процедуру приема
       TEST  AL,00100000B  ;проверяем готовность к передаче
       JZ    KETRY         ;если нет, то возвращаемся назад
;---передаем символ, принимаемый с клавиатуры
       MOV   AH,1          ;функция проверки нажатия клавиши
       INT   16H           ;прерывание клавиатуры BIOS
       JZ    KETRY         ;возврат, если не было нажатия
       MOV   AH,0          ;функция получения кода с клавиатуры
       INT   16H           ;теперь нужный символ в AL
```

```
SUB   DX,5          ;адрес регистра хранения передатчика
OUT   DX,AL         ;посылаем символ
JMP   SHORT KETRY   ;возвращаемся к началу цикла
```

7.1.7. Получение данных

Как только инициализирован коммуникационный порт [7.1.2] и установлена связь с удаленной станцией [7.1.5] - коммуникационная программа готова принимать данные. Прием данных никогда полностью не отделен от их передачи, поскольку программе может потребоваться послать сигнал XOFF (ASCII 19), чтобы остановить поток данных, если они поступают слишком быстро и она не успевает их обрабатывать. Код XON (ASCII 17) сообщает удаленной станции, что можно продолжить передачу. Отметим, что PCjr не может принимать данные во время дисковых операций; чтобы снять это ограничение, нужно применять XON и XOFF.

В зависимости от сложности протокола обмена принимаемые данные могут требовать простой или сложной обработки. Может быть получен любой код из набора управляющих кодов, приведенных в [7.1.9]. Те из них, которые являются ограничителями данных, чаще обнаруживаются при синхронном обмене. При выводе получаемых символов на экран учитывайте влияние символов перевода строки (ASCII 10), поскольку некоторые языки (включая Бейсик) автоматически вставляют их после каждого символа возврата каретки; в этом случае исключайте символы перевода строки из принимаемых данных, чтобы избежать пустых строк при выводе. На рис. 7.2 показана коммуникационная процедура, включающая также код передачи, обсуждаемый в [7.1.6].

Высокий уровень

Для коммуникационной процедуры, написанной на интерпретируемом Бейсике, время очень существенно. Обработка осуществляется очень медленно, поэтому если процедура приема неверно сконструирована, входной буфер может уже заполниться (т.е. произойдет переполнение), в то время как программа еще будет анализировать ранее полученные данные. Очевидным решением этой проблемы является применение максимально возможного размера буфера. При загрузке Бейсика размер буфера ввода устанавливается добавлением к команде ключа /C:. BASICA /C:1024 создает буфер размером в 1K и это минимальное число, необходимое для скорости обмена 1200 бод (сложным процедурам может понадобиться 4096 байт). По умолчанию используется размер буфера, равный 256 байтам, и такой буфер имеет то преимущество, что он

может быть целиком помещен в одну символьную переменную. Такой размер буфера можно использовать только при скорости обмена не более 300 бод.

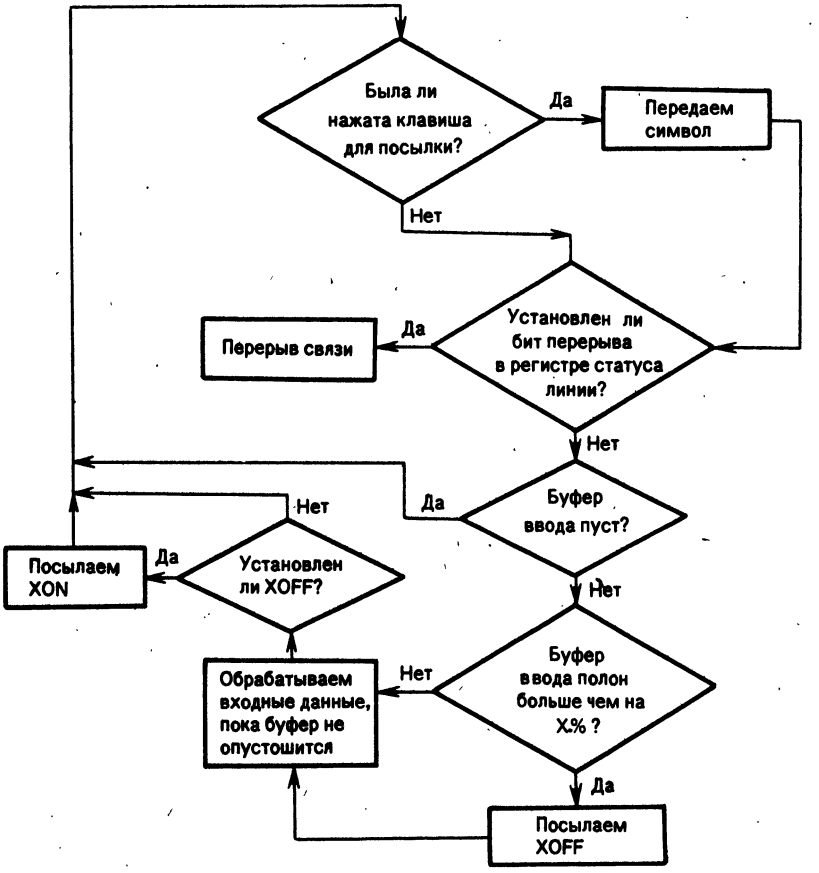


Рис. 7.2. Основная процедура коммуникации

Бейсик читает из буфера с помощью оператора INPUT$ (можно использовать также INPUT# и LINE INPUT#, но INPUT$ более гибок). Этот оператор имеет форму INPUT$(числобайт,номерфайла). Например, INPUT$(10,#1) читает 10 байт из коммуникационного канала, открытого как файл #1. Если размер буфера не превышает 256 байт, то очень удобно

читать все содержимое буфера за один раз. LOC сообщает, сколько байтов данных находится в буфере в настоящий момент. Поэтому напишите оператор INPUT$(LOC(1),#1), и в S$ будут записаны все данные с момента последнего доступа к буферу. Конечно, если LOC(1) = 0, то буфер пуст, и процедура должна ожидать, пока данные будут получены. Отметим, что EOF(1) также можно использовать для проверки состояния буфера, так как эта функция возвращает -1, если буфер пуст, и 0, если там есть хотя бы один символ.

После того как данные записаны в S$, программа должна проверить не содержатся ли там управляющие коды. Быстрее всего выполняет эту задачу функция INSTR. Напомним, что ее параметрами являются, во-первых, позиция, с которой надо вести поиск в строке, во-вторых, имя строки и, в-третьих, искомый символ (или строка). Чтобы найти символ XOFF (ASCII 19), оператор должен иметь вид INSTR(1,S$,CHR$(19)). А чтобы найти второе появление нужного управляющего символа, повторите поиск в строке, начиная с символа, следующего за позицией, в которой найден первый.

Обычно процедура ввода исключает большинство управляющих символов из принимаемых данных, с тем чтобы они нормально выглядели при выводе. Затем данные выводятся на экран, пересылаются в другое место в памяти, а иногда записываются на диск или выводятся на принтер. В процессе всей этой деятельности программа должна постоянно контролировать поступление новых данных. Если оказалось, что буфер заполняется слишком быстро, то программа может послать сигнал XOFF, останавливая поток данных. Затем, после того как полученные данные будут декодированы, можно снова разрешить передачу данных. Конечно, необходимо, чтобы протокол обмена поддерживал XON и XOFF. Обычно программы, написанные на интерпретируемом Бейсике, могут использовать XON/XOFF для установления соответствия скоростей при приеме данных, но при передаче данных такая программа часто не может достаточно быстро отреагировать на получение сигнала XOFF.

```
500 '''здесь находится процедура передачи (см. [7.1.6])
.
.
600 IF LOC(1)>100 THEN XOFF = 1: PRINT #1,CHR$(19)
                                  'посылаем XOFF при заполнении буфера
610 C$ = INPUT$(LOC(1),#1)        'читаем содержимое буфера
620 '''выделяем из данных управляющие символы
```

```
630 IF INSTR(1,C$,CHR$(19))>0 THEN 800      'получен XOFF
640 IF INSTR(1,C$,CHR$(17))>0 THEN 900      'получен XON
.
(здесь удаляются ненужные управляющие символы)
.
700 PRINT C$                                'выводим данные на экран
710 IF LOC(1) > 0 THEN 600                  'если получены данные, то читаем их
720 IF XOFF = 1 THEN XOFF = 0: PRINT #1,CHR$(17)
                                            'выключаем XOFF
730 GOTO 500                                'уход к началу процедуры передачи
.
.
800 'реакция на XOFF
.
900 'реакция на XON
```

Если функция LOF применяется к коммуникационному порту, то она возвращает информацию о количестве свободного места, оставшемся в буфере ввода. Например, если COM1 открыт как #1, то LOF(1) сообщит о количестве свободного пространства. Это может быть полезно для определения заполнения буфера. Отметим, однако, что оператор LOC возвращает позицию указателя в буфере, и это значение может быть использовано для той же цели. Например, если COM1 открыт как #3, а размер буфера ввода равен 256 байтам, то до тех пор, пока LOC(3) не вернет 256, буфер не будет полным.

Средний уровень

Функция 2 прерывания 14H BIOS ожидает символ из последовательного порта, помещает его в AL при получении и затем возвращается в программу. При входе надо поместить номер порта (0-1) в DX. При возврате AX равен нулю, если не было ошибки. Если AH не равен 0, то может быть возвращен байт статуса, в котором имеют значение только 5 бит. Это следующие биты:

бит	1	ошибка переполнения (новый символ поступил раньше, чем был удален старый)
	2	ошибка четности (вероятно, из-за проблем в линии)
	3	ошибка оформления (стартовый или стоп-биты неверны)
	4	обнаружен перерыв (получена длинная строка битов 0)
	5	ошибка тайм-аута (не получен сигнал DSR)

MS-DOS также предоставляет коммуникационную функцию для приема одного символа - функцию 3 прерывания 21H. Она ожидает символ из COM1 и помещает его в AL. Отметим, что при этом нет функции инициализации порта, которую надо делать с помощью процедуры BIOS или непосредственно, как показано в [7.1.2]. По умолчанию порт инициализируется со значениями 2400 бод, нет контроля четности, один стоп-бит и 8 бит на символ. Эта функция не имеет никаких преимуществ по сравнению с функцией BIOS и не возвращает информации о статусе.

Низкий уровень

При получении данных без использования коммуникационного прерывания [7.1.8] программа должна постоянно проверять регистр статуса линии, адрес порта которого на 5 больше базового адреса используемого коммуникационного адаптера. Бит 0 этого регистра будет равен нулю, пока не будет получен символ в регистр данных приемника. Когда бит 0 становится равным 1, надо немедленно считать его из регистра, с тем чтобы на него не наложился следующий принимаемый символ. После того как символ считан, бит 0 опять становится равным 0 и остается таковым, пока не прибудет новый символ.

Хотя об этом еще не упоминалось, но вы должны знать, что коммуникационные процедуры обычно создают *циклический буфер* для сбора поступающих символов. Циклические буфера обсуждались в [3.1.1]. Вы должны также знать, что если поступающие данные подавать на экран со скоростью 1200 бод, то процедура сдвига экрана BIOS [4.5.1] не будет успевать и произойдет переполнение. Простое решение этих проблем состоит в использовании коммуникационного прерывания, как объяснено в [7.1.8].

В следующем примере частично повторяется содержимое предыдущего параграфа, относящегося к передаче символов. Как и в том случае, код начинается с бесконечного цикла. Объедините эти две процедуры с процедурами инициализации из [7.1.2] и [7.1.5] для создания законченной процедуры ввода / вывода через коммуникационный канал.

```
KETRY: MOV DX,BASADR    ;базовый адрес
       ADD DX,5         ;указываем на регистр статуса линии
       IN  AL,DX        ;получаем байт статуса
       TEST AL,00011110B ;проверяем на ошибку
       JNZ  ERRROUT     ;если да, то на обработку ошибки
       TEST AL,00000001B ;проверяем, получены ли данные
       JNZ  RECVE       ;на процедуру приема данных
       TEST AL,00100000B ;проверяем готовность к передаче
```

```
        JZ    KETRY          ;если нет, то к началу цикла
.
(здесь расположена процедура передачи - см. [7.1.6])
.
;---получаем данные и выводим их на экран
RECVE: MOV DX,BASE_ADDRESS   ;базовый адрес
        IN    AL,DX          ;читаем полученный символ
        CMP   AL,19          ;проверка на XOFF
        JE    XOFF_ROUTINE
.
                             ; (и т.д.)
.
        MOV DL,AL            ;готовим символ для вывода на экран
        MOV AH,2             ;функция вывода символа
        INT 21H              ;выводим его
        JMP SHORT KETRY;возвращаемся к началу цикла
```

7.1.8. Посылка/получение данных с помощью коммуникационного прерывания

Тщательно продуманная коммуникационная программа имеет слишком много обязанностей, чтобы посвятить себя целиком вводу/выводу. Поступающие данные должны анализироваться, передаваемые данные должны собираться, а большие блоки данных могут записываться на диск или считываться с него. Коммуникационное прерывание позволяет программе не тратить на ввод/вывод больше времени, чем он того требует. Например, после установки прерывания управление передается процедуре передачи данных только в том случае, если регистр хранения передатчика пуст, и возвращается программе, как только послан байт данных, позволяя ей продолжать свою работу до тех пор, пока регистр хранения передатчика не будет снова готов. Не забудьте ознакомиться с обсуждением прерываний в [1.2.3], прежде чем продолжить чтение.

IBM PC отводит два аппаратных прерывания для коммуникационных каналов, номера 3 (COM1) и 4 (COM2). Отметим, что у PCjr встроенный модем имеет номер 3, а COM1 - номер 4. Микросхема UART 8250 допускает 4 класса прерываний для каждого канала, используя следующие двоичные кодовые числа:

00 изменение в регистре статуса модема
01 регистр хранения передатчика пуст
10 получены данные
11 ошибка приема, или получено условие перерыва

Эти коды содержатся в битах 2-1 регистра идентификации прерывания, адрес порта которого на 2 больше, чем базовый адрес используемого коммуникационного адаптера. Бит 0 этого регистра устанавливается при возникновении прерывания, а остальные биты не используются и всегда равны 0.

Чтобы выбрать одно или более прерываний, надо запрограммировать регистр разрешения прерывания, адрес которого на 1 больше базового адреса. Значения его битов могут быть такими:

бит 0 1 = прерывание при получении данных
1 1 = прерывание, когда регистр хранения передатчика пуст
2 1 = прерывание при ошибке приема данных
3 1 = прерывание при изменении регистра статуса модема
7-4 не используются, всегда 0

Когда одно из этих событий происходит, инициируется аппаратное прерывание, возникающее в микросхеме обработки прерываний 8259 по каналу 3 для COM1 и по каналу 4 для COM2. Процедура обработки прерываний передает управление тому коду, на который указывает соответствующий вектор прерывания. Поскольку это аппаратное прерывание, то оно может быть маскировано (см. [1.2.2]). Помните, что процедура обработки прерывания должна завершаться стандартным кодом выхода из аппаратного прерывания MOV AL,20H/OUT 20H,AL. На рис. 7.3 показано коммуникационное прерывание.

Одновременно возможно любое число типов прерывания. Но если разрешен более чем один тип, процедура обработки прерывания должна сама определять, какой из типов прерывания произошел, проверяя регистр идентификации прерывания. Одновременно могут происходить более чем одно прерывание, поэтому бит 0 регистра идентификации сообщает о том, что поступило еще одно прерывание. Когда два или более прерываний поступают в один и тот же момент, то они обрабатываются в порядке, указанном в приведенной ниже таблице. Добавочные прерывания должны быть обработаны до завершения процедуры обработки прерывания. Условия предшествующих прерываний "отменяются" с помощью действий, предлагаемых в правом столбце следующей таблицы:

Код	Тип	Действия для "сброса"
11	ошибка или перерыв	чтение регистра статуса линии
10	получены данные	чтение регистра приемника данных
01	передатчик готов	вывод символа в регистр хранения передатчика
00	изменение статуса модема	чтение регистра статуса модема

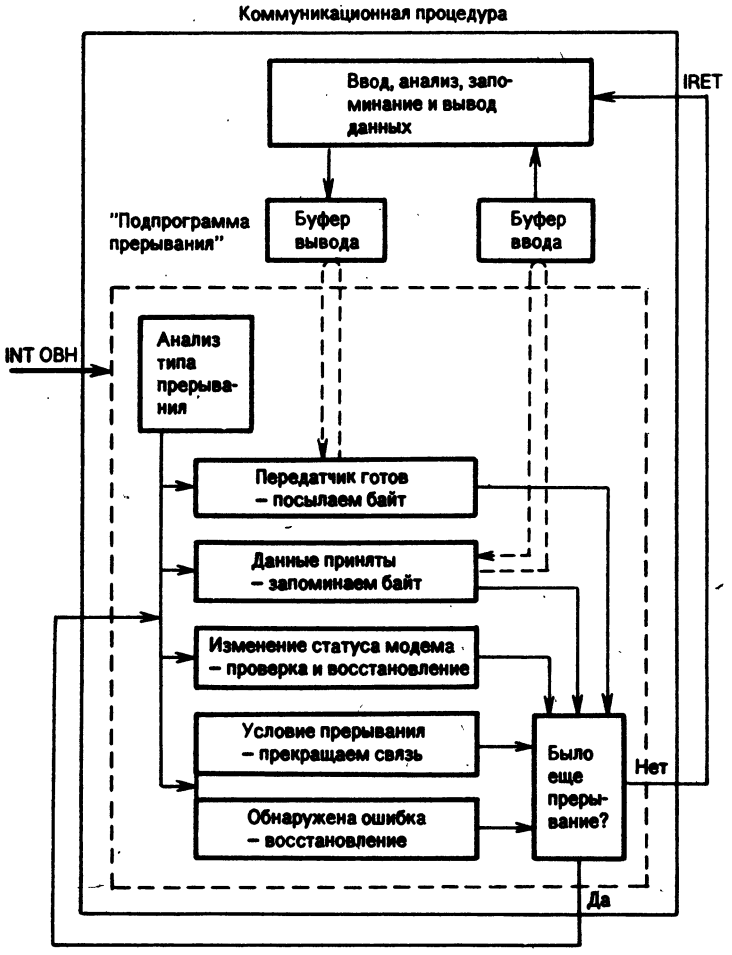


Рис. 7.3. Коммуникационное прерывание

Низкий уровень

Приведем общую форму программы, обрабатывающей коммуникационные прерывания:

```
;---установка вектора коммуникационного прерывания
        PUSH DS                 ;сохраняем DS
        MOV  DX,OFFSET IOIN ;DS:DX указывают на процедуру
        MOV  AX,SEG IOIN
        MOV  DS,AX
        MOV  AL,0BH             ;номер вектора для COM1
        MOV  AH,25H             ;функция изменения вектора
        INT  21H                ;меняем вектор прерывания
;---инициализация регистра разрешения прерывания (COM1)
        MOV  AX,40H             ;DS указывает на данные BIOS
        MOV  DS,AX
        MOV  DX,DS:[0]          ;получаем базовый адрес COM1
        INC  DX                 ;указываем на регистр разрешения
        MOV  AL,3               ;прерываний и разрешаем прерывания
        OUT  DX,AL              ;приема и передачи
        POP  DS                 ;восстанавливаем регистр
;---процедура обработки прерывания - сначала определяем его тип
IOIN    PROC FAR
NEXIN:  MOV  DX,BASADR          ;базовый адрес
        INC  DX                 ;указываем на регистр идентификации
        INC  DX                 ;прерывания
        IN   AL,DX              ;читаем его значение
        TEST AL,10B             ;это прерывание передатчика?
        JNZ  TRANSMIT           ;если да, то на передачу
RECEIVE:                        ;иначе - на прием
.
.
        JMP  SHORT ANOT         ;проверяем, нет ли другого прерывания
TRANSMIT:                       ;начинаем процедуру передачи символа
.
.
;---перед выходом проверяем, нет ли другого прерывания
ANOT:   MOV  DX,BASADR          ;базовый адрес
        INC  DX                 ;указываем на регистр идентификации
        INC  DX                 ;прерывания
        IN   AL,DX              ;читаем его значение
        TEST AL,1               ;проверяем бит 1
        JNZ  NEXT_INT           ;если он установлен, то на начало
        MOV  AL,20H             ;иначе, код завершения аппаратного
```

```
        OUT  20H,AL          ;прерывания
        IRET
IO_INT  ENDP
```

7.1.9. Сводка управляющих кодов, используемых при коммуникации

В таблице 7.1 приведены 32 управляющих кода ASCII, которые используются при коммуникации, а также при работе принтера и других устройств. Добавлен также код ASCII 127 - DEL ("забой"), который является управляющим кодом, хотя его и нельзя выдать с клавиатуры с помощью сочетания Ctrl + клавиша. Применение некоторых кодов, таких, как возврат каретки, инвариантно. Но большинство других управляющих кодов имеет широкий диапазон интерпретации во многом из-за несовместимости оборудования.

7.2. Создание драйвера устройства

Драйвер устройства - это специальная программа, которая управляет обменом с периферийным устройством, таким, как принтер или дисковый накопитель. Поскольку параметры этих периферийных устройств меняются в зависимости от производителя, то разным пользователям программы может потребоваться дюжина различных драйверов, чтобы он мог работать на имеющемся у него оборудовании. Имеется 4 способа включения драйверов устройств в программу.

1. Можно поместить код для всех драйверов непосредственно в программу. Например, чтобы поддерживать различные принтеры, можно создать таблицу управляющих последовательностей и искать в ней нужный код каждый раз, когда он потребуется. Этот подход тратит много памяти и может быть достаточно медленным.

2. Создать ряд драйверов устройств и потребовать, чтобы программа загружала необходимый в качестве оверлея (т.е. помещать его в область программы, специально оставленную для этой цели (см. [1.3.5])).

3. Создать драйвер устройства как отдельную программу, которая указывается в командном файле, выполняемом при загрузке системы. Программа запускается и устанавливает драйвер устройства как программу обработки прерывания. После этого программа завершается, но остается резидентной в памяти, как объяс-

Таблица 7.1

Номер кода 10-й	16-й	ASCII Символ	Комбинаций с Ctrl	Мнемоника	Назначение
00	00	(null)	^@	NUL	Символ–разделитель (не имеющий значения; необходим для задержек).
01	01		^A	SOH	Начало заголовка. Начинает передачу блока данных или нового файла.
02	02	●	2^B	STX	Начало текста. Отмечает начало текста, следующего за заголовком данных.
03	03	♥	^C	ETX	Конец текста. Может отмечать начало данных при проверке ошибок.
04	04	♦	^D	EOT	Конец передачи. Код остановки, но иногда он просто отмечает конец файла.
05	05	♣	^E	ENQ	Запрос. Запрашивает статусную информацию у отдаленной станции.
06	06	♠	^F	ACK	Подтверждение. Подтверждает успешный обмен между станциями.
07	07		^G	BEL	Звонок. Инициирует звонок, чтобы привлечь внимание.
08	08	■	^H	BS	Возврат на шаг.
09	09	○	^I	HT	Горизонтальная табуляция.
10	0A	■	^J	LF	Перевод строки.
11	0B	♂	^K	VT	Вертикальная табуляция.
12	0C	♀	^L	FF	Перевод формата.
13	0D	♪	^M	CR	Возврат каретки.
14	0E	♫	^N	SO	Сдвиг выключен. Переключает набор символов.
15	0F	☼	^O	SI	Сдвиг включен. Переключает набор символов.
16	10	►	^P	DLE	Data Link Escape. Модифицирует значение следующих символов (аналогично Esc).
17	11	◄	^Q	DC1	Управление устройством 1. Используется как сигнал XON удаленной станции на передачу.
18	12	↕	^R	DC2	Управление устройством 2. Сигнал переключения общего назначения.
19	13	‼	^S	DC3	Управление устройством 3. Используется как сигнал XOFF удаленной станции для прекращения передачи.
20	14	¶	^T	DC4	Управление устройством 4. Сигнал переключения общего назначения.
21	15	§	^U	NAK	Отрицание. Передача неуспешна.
22	16	▬	V	SYN	Промежуток синхронизации. Используется между блоками при синхронной связи.
23	17	↨	^W	TB	Конец блока передачи. Вариант ETX.
24	18	↑	^X	CAN	Отмена. Обычно сигнализирует об ошибке передачи.
25	19	↓	Y	EM	Конец среды. Сигнализирует о физическом конце источника данных.
26	1A	→	^Z	SUB	Подстановка. Заменяет ошибочные символы, а также те, которые невозможно изобразить.
27	1B	←	^[	ESC	Отмечает последующие символы как управляющую последовательность.
28	1C	∟	^/	FS	Разделитель файлов. Отмечает логическую границу между файлами.
29	1D	↔	^]	GS	Разделитель групп. Отмечает логическую границу между группами данных.
30	1E	▲	^^	RS	Разделитель записей. Отмечает логическую границу между записями данных.
31	1F	▼	^_	US	Разделитель объектов. Отмечает логическую границу между объектами данных.
127	7F	⌂	нет	DEL	Забой. Удаляет другие символы.

нено в [1.3.4]. Впоследствии наша программа использует этот драйвер через вектор прерывания.

4. Создать полноценный драйвер устройства, который будет загружаться при старте с помощью файла CONFIG.SYS. MS-DOS поддерживает такой тип драйверов устройств и однажды загруженный он может использовать все возможности команд DOS, включая проверку ошибок. Специальная команда IOCTL (контроль ввода/вывода) позволяет программе узнать статус драйвера и послать ему управляющую строку помимо обычного потока данных.

Первые три способа легко реализовать с помощью информации, приведенной в остальных частях данной книги. Устанавливаемые драйверы устройств очень сложны, зато когда они имеются - у вас есть хороший помощник. В этом случае система будет работать с устройством настолько же тесно, как с клавиатурой или дисковым накопителем. Устройству может быть присвоено имя, например, SERIALPR для последовательного принтера, и затем оно может быть открыто для доступа из любого языка. В Бейсике оператор OPEN "SERIALPR" FOR OUTPUT AS #2 подготовит последовательный принтер для вывода. В языке ассемблера вы сможете получить доступ к принтеру как с помощью метода управляющего блока файла, так и с помощью метода дескриптора файла, включая очень мощную функцию IOCTL. При этом пользователь имеет возможность доступа к устройству на уровне операционной системы и может просто ввести команду COPY A:MYFILE SERIALPR:, чтобы скопировать содержимое файла на принтер.

Устанавливаемые драйверы устройств могут быть написаны только на языке ассемблера и обслуживать два типа устройств: символьные и блочные. Эти имена описывают единицы, которыми устройство обрабатывает данные. Обычно драйверы блочных устройств обслуживают дисковые накопители, а драйверы символьных - все остальное, начиная от последовательных принтеров и кончая роботами. Блочные устройства обмениваются блоками данных, поэтому они занимаются накоплением данных. Символьные устройства обмениваются данными побайтно, поэтому они лучше подходят для управляющих устройств, а также для устройств, которые не могут обеспечить высокую скорость обмена данными. Драйверы блочных устройств очень сложны и здесь нет достаточно места, чтобы объяснить их структуру. Редко кому требуется написать такой драйвер. Техническое руководство по MS-DOS предоставляет всю необходимую информацию и содержит полный пример драйвера виртуального диска в оперативной памяти. Вы можете просмотреть эту информацию, после того как изучите обсуждение драйверов символьных устройств, приведенное здесь.

Устанавливаемые драйверы устройств беспощадны к ошибкам программистов. Поскольку драйверы автоматически загружаются системой при загрузке, то невозможно использовать отладчики для выявления причин неполадок. Поэтому будьте предельно внимательны при их написании.

Программа драйвера устройства разбивается на три части, каждая из которых обсуждается отдельно в следующих параграфах. Это (1) заголовок драйвера, который именует устройство и содержит информацию об остальных частях драйвера, (2) стратегия драйвера, которая хранит информацию об области данных, создаваемой MS-DOS, которая называется заголовком запроса, и (3) обработчик прерывания устройства, который и содержит код, управляющий устройством.

7.2.1. Создание заголовка драйвера

Драйверы устройств должны создаваться в виде COM-файлов [1.3.6]. Однако они не являются настоящими программами, поскольку у них отсутствует префикс программного сегмента. Чтобы добиться этого, не надо включать оператор ORG 100H в начале программы, как это делается для COM-файлов. Либо запишите ORG 0, либо вообще ничего не пишите. Драйвер должен быть описан как далекая (far) процедура, как и в любой программе. В следующем примере приведен начальный код для драйвера устройства с именем DEVICE12. Оно заменяет стандартное устройство AUX, используемое MS-DOS, принимая вывод функции 4 прерывания 21H. Весь драйвер устройства состоит из кода этого раздела вместе с кодом, приведенном в следующих двух разделах; поместите их подряд один за другим, чтобы получить полную программу.

Драйвер устройства должен начинаться с заголовка драйвера. Он имеет длину 18 байт, разделенных на 5 полей. Первое поле (DD) всегда содержит значение -1 (FFFFFFFFH), и когда MS-DOS загружает драйвер, то оно заменяется на стартовый адрес следующего драйвера. Таким образом, система может искать следующий драйвер по цепочке. У последнего загруженного драйвера в этом поле остается значение -1.

Второе поле это байт атрибутов драйвера. Имеют значение только 7 бит:

бит 15 1 = символьное устройство, 0 = блочное устройство
14 1 = поддерживает IOCTL, 0 = не поддерживает IOCTL
13 1 = формат блоков IBM, 0 = другой формат блоков
3 1 = часы, 0 = не часы
2 1 = нулевое устройство, 0 = не нулевое устройство

1 1 = устройство стандартного вывода, 0 = нет
0 1 = устройство стандартного ввода, 0 = нет

Обычно установлен только бит 15, или биты 15 и 14, если устройство поддерживает IOCTL (как обсуждается в [7.2.4]). Бит 13 устанавливается только для блочных устройств. Остальные биты используются для замены устройств, включаемых MS-DOS по умолчанию (устройствами стандартного ввода и вывода являются клавиатура и видеодисплей; устройство часов объединяет часы реального времени с часами времени суток BIOS; а нулевое устройство (NULL) - это псевдоустройство, используемое для тестовых целей).

Третье и четвертое поля содержат смещения для процедур стратегии и обработки прерывания, которые будут рассмотрены в следующих параграфах. Наконец, последнее поле содержит имя устройства. Имя может содержать до 8 символов, и оно должно быть выравнено по левому краю с завершающими пробелами. Для замены существующих в DOS устройств, таких, как LPT1 или COM1, надо указать то же имя устройства, как в данном примере.

Низкий уровень

В данном примере создается драйвер для последовательного устройства. "DEV12" - имя файла, который должен быть указан в файле конфигурации сиситемы, чтобы этот драйвер был загружен. В байте атрибутов установлен только бит 15, указывая, что это символьное устройство и что оно не поддерживает IOCTL. DEV_STRATEGY и DEV_INTERRUPT - имена процедур, обсуждаемых в следующих параграфах. Устройство названо AUX, с тем чтобы заменить обычное устройство MS-DOS с этим именем. Это позволяет очень просто обращаться к этому устройству, поскольку система имеет предопределенный номер файла для обращения к устройству AUX (последовательному). В пример включен начальный код для драйвера, определяющий его как COM-программу.

```
CSEG    SEGMENT PUBLIC 'CODE'  ;устанавливаем кодовый сегмент
        ORG 0                  ;эта строка необязательна
        ASSUME CS:CSEG,DS:CSEG,ES:CSEG
DEV12   PROC FAR               ;драйвер это далекая процедура
        DD   0FFFFFFFFH        ;адрес следующего драйвера
        DW   8000H             ;байт атрибутов
        DW   DEV_STATEGY       ;адрес процедуры стратегии
        DW   DEV_INTERRUPT     ;адрес процедуры прерывания
        DB   'AUX      '       ;имя устройства (дополненное пробелами)
```

7.2.2. Создание стратегии устройства

Для процедуры стратегии устройства необходимо только пять строк. Когда система загружает устройство, она создает блок данных, называемый *заголовком запроса*. Он имеет две функции. Во-первых, он служит областью данных для внутренних операций системы. Во-вторых, что более важно, заголовок запроса служит областью, через которую происходит обмен информацией между драйвером и вызывающей его программой. Например, когда драйвер выводит данные, адрес данных он получает через заголовок запроса. Когда же драйвер завершает свою работу, он устанавливает в заголовке запроса байт статуса, который доступен вызывающей программе, тем самым давая возможность ей узнать об ошибке.

MS-DOS создает заголовок запроса при установке драйвера устройства (когда система загружается). В этот момент (и только один раз) выполняется процедура стратегии устройства. При этом ES:BX указывают на вновь созданный заголовок запроса и процедуре нужно просто скопировать их, чтобы впоследствии он мог быть обнаружен при обращении к драйверу. Адреса смещения и сегмента заголовка помещаются в две переменные. В следующем разделе вы увидите, что при обращении к драйверу, первое, что он делает - восстанавливает значения ES:BX, чтобы можно было получить информацию из заголовка запроса.

Размер заголовка запроса может меняться, в зависимости от типа запроса к драйверу (например, инициализация, вывод данных или возврат статуса). Однако первые 13 байт заголовка всегда одни и те же. Приведем их формат:

1. Длина заголовка запроса (DB).
2. Код устройства (DB). Определяет номер для блочных устройств.
3. Код команды (DB). Здесь хранится номер последней посланной драйверу команды. Эти коды перечислены в [7.2.3].
4. Статус (DW). Статус устанавливается каждый раз при вызове драйвера. Если установлен бит 15, то в младших восьми битах находится код ошибки. Коды ошибок перечислены в [7.2.3].
5. Резервная область (8 байт). Используется MS-DOS.
6. Данные, необходимые для работы драйвера (переменной длины).

Низкий уровень

Ниже представлены 5 строк процедуры стратегии устройства. Отмечаем, что две словные переменные, хранящие значения ES и BX, следуют за инструкцией RET, как и положено в формате COM.

```
DEV_STRATEGY: MOV  CS:KEEP_ES,ES
              MOV  CS:KEEP_BX,BX
              RET
KEEP_CS       DW   ?
KEEP_BX       DW   ?
```

7.2.3. Создание обработчика прерывания устройства

Драйвер устройства начинается с двух порций кода, описанных ранее. За ними должна следовать соответствующая процедура обработки прерывания. На самом деле неверно называть эту процедуру процедурой обработки прерывания, так как она вовсе не обслуживает прерывание и завершается обычной инструкцией RET.

Имеется 13 типов функций, которые может выполнять устанавливаемый драйвер устройства. Когда драйвер вызывается функцией DOS (скажем, функцией 3FH прерывания 21H, которая читает данные из файла или устройства), то функция помещает кодовый номер от 1 до 13 в однобайтовое поле по смещению 2 в заголовке запроса (для ввода - кодовый номер 5). Затем управление передается процедуре обработки прерывания драйвера, адрес которой определяется при просмотре заголовка драйвера [7.2.1]. Эта процедура в первую очередь восстанавливает ES:BX, с тем чтобы они указывали на заголовок запроса, а затем читает кодовый номер команды. По этому коду процедура обработки прерывания вызывает нужную процедуру, которая выполнит требуемую функцию. Процедура ищется с помощью 13-словной таблицы, содержащей смещения для 13 типов функций. Функции всегда перечисляются в следующем порядке:

1. INITIALIZE (инициализация)
2. CHECK_MEDIA (проверка носителя)
3. MAKE_BPB (блок параметров диска)
4. IOCTL_IN (контроль ввода/вывода - вход)
5. INPUT_DATA (ввод данных)
6. NONDESTRUCT_IN
7. INPUT_STATUS (статус ввода)
8. CLEAR_INPUT (очистка ввода)

9. OUTPUT_DATA (вывод данных)
10. OUTPUT_VERIFY (проверка вывода)
11. OUTPUT_STATUS (статус вывода)
12. CLEAR_OUTPUT (очистка вывода)
13. IOCTL_OUT (контроль ввода/вывода - выход)

После выполнения задания процедура обработки прерывания завершается инструкцией RET, и управление возвращается в вызывающую программу. Драйвер устройства может включать код для обработки только некоторых функций в зависимости от устройства и требуемой степени контроля ошибок и управления устройством. Номера функций, для которых не написаны процедуры, должны завершаться выходом из драйвера без выполнения чего-либо. В этом случае надо только перед выходом установить биты 15, 8, 1 и 0 в заголовке запроса, чтобы информировать вызывающую программу, что была затребована несуществующая функция (бит 15 индицирует ошибку, бит 8 показывает, что драйвер работает нормально, а биты 0 и 1 дают код ошибки 3, что соответствует "неизвестной команде").

Но одна функция должна присутствовать во всех драйверах устройств - функция 1 (инициализация). Эта функция автоматически выполняется только при загрузке драйвера. Одна из важных задач, выполняемая этой процедурой, состоит в установке адреса конца драйвера в четырех байтах, начинающихся со смещения 14 в заголовке запроса. В приведенном ниже примере конец программы отмечен меткой eop:. Кроме этой задачи, процедура инициализации должна также выполнить всю необходимую для данного устройства инициализацию. На рис. 7.4 показана структура драйвера устройства.

Какие из оставшихся 12 функций будут включены в драйвер устройства зависит от того, что должен делать драйвер. Некоторые, такие, как CHECK_MEDIA и MAKE_BPB, относятся только к блочным устройствам (они устанавливают тип диска, размер секторов и т.д.). Для символьных устройств наиболее важными являются две функции: INPUT_DATA и OUTPUT_DATA (отметим, что эти имена несущественны - важна позиция в таблице функций, которая неизменна). В обоих случаях заголовок запроса имеет следующую структуру:

13 байт	стандартный формат заголовка запроса
1 байт	байт описания среды (только для блочных устройств)
4 байта	смещение/сегмент буфера обмена данных
2 байта	число байтов, которое надо передать
2 байта	стартовый номер сектора (только для блочных)

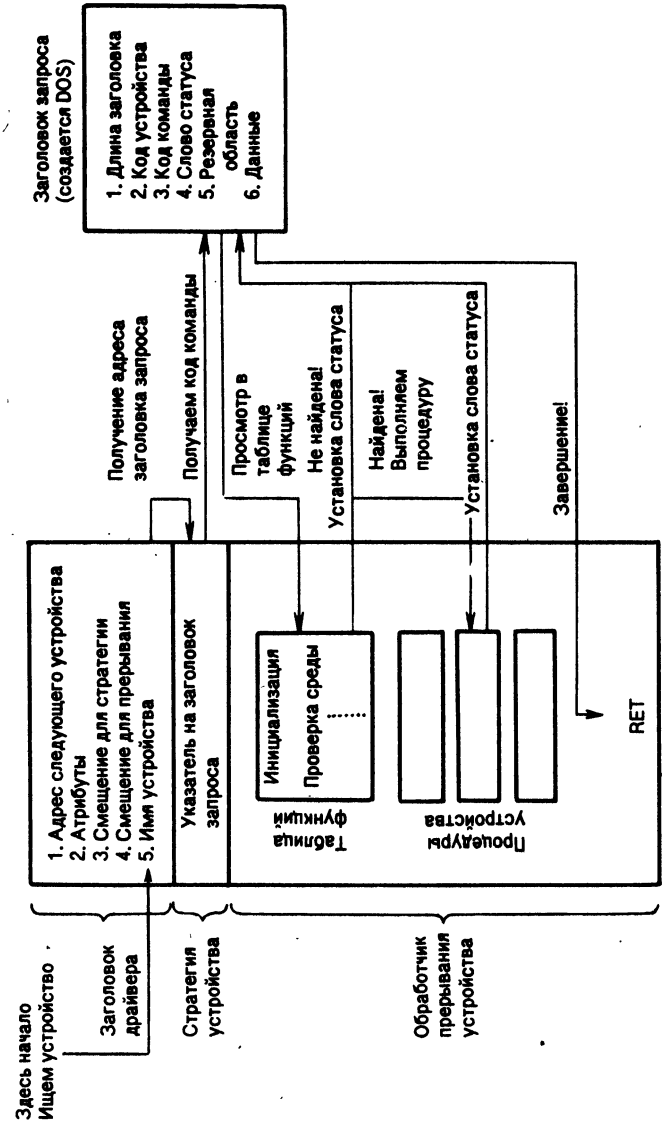


Рис. 7.4. Драйвер символьного устройства

В приведенном ниже примере используется функция вывода. Процедура, выполняющая вывод, получает из заголовка запроса адрес буфера, в котором находятся выводимые данные (смещение 14). Она также считывает число байтов, которое надо вывести (смещение 18). Когда процедура завершит вывод данных, она установит слово статуса в заголовке запроса (смещение 3) и возвратит управление. Если операция завершится успешно, то надо установить бит 8 слова статуса. Другие возможности будут обсуждены позднее.

Низкий уровень

В данном примере приведена общая форма процедуры обработки прерывания, не включая реального кода, управляющего устройством.

```
;---инициализация обработчика прерывания устройства
DEV_INTERRUPT: PUSH ES      ;сохраняем регистры
PUSH DS
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
PUSH DI
PUSH BP
MOV  AX,CS:KEEP_ES          ;ES:BX указывают на заголовок запроса
MOV  ES,AX
MOV  BX,CS:KEEP_BX
MOV  AL,ES:[BX] + 2         ;получаем код команды из заголовка
SHL  AL,1                   ;умножаем на 2 (так как таблица словная)
SUB  AH,AH                  ;обнуляем AH
LEA  DI,FUNCTIONS           ;DI указывает на смещение до таблицы
ADD  DI,AX                  ;добавляем смещение в таблице
JMP  WORD PTR [DI]          ;переходим на адрес из таблицы
FUNCTIONS LABEL WORD        ;это таблица функций
DW   INITIALIZE
DW   CHECK_MEDIA
DW   MAKE_BPB
DW   IOCTL_IN
DW   INPUT_DATA
DW   NONDESTRUCT_IN
DW   INPUT_STATUS
DW   CLEAR_INPUT
```

```
DW    OUTPUT_DATA
DW    OUTPUT_VERIFY
DW    OUTPUT_STATUS
DW    CLEAR_OUTPUT
DW    IOCTL_OUT
;---выход из драйвера, если функция не поддерживается
CHECK_MEDIA:
MAKE_BPB:
IOCTL_IN:
INPUT_DATA:
NONDESTRUCT_IN:
INPUT_STATUS:
CLEAR_INPUT:
OUTPUT_VERIFY:
OUTPUT_STATUS:
CLEAR_OUTPUT:
IOCTL_OUT:
OR    ES:WORD PTR [BX] + 3,8103H    ;модифицируем статус
JMP   QUIT                          ;на завершение программы
;---процедуры для двух поддерживаемых кодов
INITIALIZE: LEA AX,E_O_P            ;смещение конца программы в AX
MOV ES:WORD PTR [BX] + 14,AX ;помещаем его в заголовок
MOV ES:WORD PTR [BX] + 16,CS ;помещаем сегмент конца программы
.
(здесь идет инициализация устройства)
.
JMP QUIT
OUTPUT_DATA: MOV CL,ES:[BX] + 18  ;получаем число символов
CBW  CX                           ;CX используем как счетчик
MOV  AX,ES:[BX] + 16              ;получаем адрес буфера данных
MOV  DS,AX
MOV DX,ES:[BX] + 14
.
(здесь идут операции по выводу)
.
JMP   QUIT
;---выходим, модифицируя байт статуса в заголовке запроса
QUIT: OR ES:WORD PTR [BX] + 3,100H ;устанавливаем бит 8
POP   BP                          ;восстанавливаем регистры
POP   DI
POP   SI
POP   DX
POP   CX
POP   BX
POP   AX
```

```
POP  DS
POP  ES
RET
E_O_P:                          ;метка конца программы
DEVICE12 ENDP
CSEG ENDS
END DEVICE12
```

Перед возвратом драйвер устанавливает слово статуса в заголовке запроса. В предыдущем примере это делается в двух местах, в зависимости от того, вызывалась функция, обеспечиваемая драйвером или нет. Эти строки выглядят так: OR ES:WORD PTR [BX]+3,XXXXH. Значения битов XXXX следующие:

биты	0-7	код ошибки (если бит 15 = 1)
	8	устанавливается в 1, когда функция завершена
	9	устанавливается в 1, когда драйвер занят
	10-14	зарезервированы MS-DOS
	15	устанавливается при возникновении ошибки

В случае установления бита 15, индицирующего ошибку, младший байт этого слова содержит следующие коды ошибок:

0	попытка записи на защищенное от записи устройство
1	неизвестное устройство
2	устройство не готово
3	неизвестная команда
4	ошибка проверки по контрольной сумме
5	неверная длина запроса к устройству
6	ошибка поиска
7	неизвестный носитель
8	сектор не найден
9	нет бумаги в принтере
A	ошибка записи
B	ошибка чтения
C	общая ошибка

7.2.4. Доступ к драйверу устройства

Драйвер устройства устанавливается путем включения имени готовой программы в файл конфигурации системы. Для установки пробной программы поместите в файл CONFIG.SYS строку DEVICE = DEVICE12.COM. Затем перезагрузите систему для установки драйвера. Если машина не будет загружаться, то скорее всего имеется ошибка в коде инициализации драйвера.

После того как драйвер установлен, для доступа к нему пользуйтесь обычными функциями прерывания 21H MS-DOS. Какие понадобятся функции зависит от того, заменяет ли устройство стандартное устройство DOS (как в приведенном примере) или оно добавляется как совершенно новое. Для замены стандартного последовательного устройства назовите драйвер AUX, после чего функции 3 ([7.1.7]) и 4 ([7.1.6]) прерывания 21H будут осуществлять соответственно ввод и вывод. Если устройство параллельное, то назовите его PRN, после чего функция 5 [6.3.1] будет выводить данные на принтер. Другой возможностью является применение функции 3FH ([5.4.4]) для ввода и 40H ([5.4.3]) для вывода. В этом случае используйте номер файла 3 для последовательного устройства и 4 для параллельного. Напоминаем, что при работе с предопределенными номерами файлов нет необходимости открывать устройство.

Если устройство не заменяет одно из стандартных устройств MS-DOS (т.е. если оно не названо одним из резервных слов, таким, как PRN, AUX и т.д.), то вы можете открыть устройство одной из функций для открытия файла. Вы можете использовать как метод доступа с помощью управляющего блока файла, так и метод дескриптора файла, хотя последний предпочтительнее. Чтобы быть уверенным, что вы по ошибке не откроете дисковый файл, поместите номер файла в BX, 0 - в AL, после чего выполните функцию 44H прерывания 21H. Это функция IOCTL, и если бит 7 значения, возвращаемого в DL, установлен, то драйвер устройства загружен.

IOCTL требует, чтобы в байте атрибутов драйвера была соответствующая установка битов и чтобы по крайней мере основы процедуры обработки IOCTL имелись в процедуре обработчика прерывания драйвера. Функция IOCTL имеет 8 подфункций, пронумерованных от 0 до 7, при этом соответствующий кодовый номер помещается в AL при вызове функции:

0 возвратить информацию об устройстве в DX

1 установить информацию об устройстве, используя DL (DH = 0)

2 считать CX байт от драйвера устройства через управляющий канал и поместить их, начиная с DS:DX

3 записать CX байт в драйвер устройства через управляющий канал, взяв их начиная с DS:DX

4 то же, что и 2, но использовать номер накопителя в BL, где 0 = накопитель по умолчанию, 1 = A и т.д.

5 то же, что и 3, но использовать номер накопителя как в 4

6 получить статус ввода

7 получить статус вывода

В ответ возвращается различная информация, в зависимости от того, какая функция вызвана. Для подфункций 0 и 1 значение битов регистра DX следующее (при условии, что бит 7 = 1, что означает, что доступ получен к устройству, а не к файлу):

0	1 = устройство консольного ввода
1	1 = устройство консольного вывода
2	1 = нулевое устройство
3	1 = устройство часы
4	резерв
5	1 = нет проверки на Ctrl-Z, 0 = есть проверка на Ctrl
6	1 = не конец файла, 0 = конец файла
7	1 = устройство, 0 = дисковый файл
8-13	резерв
14	1 = если можно использовать подфункции 2 и 3, 0 = нельзя
15	резерв

Подфункции 2-5 позволяют программе и устройству обмениваться произвольными управляющими строками. Это дает возможность передавать управляющие сообщения отдельно от основного потока данных, что существенно упрощает дело. При возврате AX будет содержать число переданных байтов. Подфункции 6-7 позволяют программе проверить, готово ли устройство для ввода или вывода. Для устройств в AL возвращается FF, если оно готово, и 0, если нет. Если же эти подфункции применить к открытому файлу (бит 7 = 0), в AL возвращается FF до тех пор, пока не будет достигнут конец файла.

Отметим, что в Бейсике 3.0 добавлены операторы IOCTL и IOCTL$. Они позволяют бейсиковской программе соответственно посылать и принимать управляющие строки от драйвера устройства, которое было предварительно открыто оператором OPEN. Выходная строка должна быть заключена в кавычки, как в IOCTL #3,"...". Подобным образом A$ = IOCTL$(3) принимает информацию о статусе через IOCTL.

7.2.5. Обнаружение и анализ ошибок устройства

Устройства могут ошибаться по одной из трех причин. Устройство может быть физически повреждено или находиться не в том состоянии. Программное обеспечение, управляющее устройством, может работать неправильно, и, наконец, программа может послать устройству недопустимый запрос (например, попытка писать на накопитель, где находится дискета, защищенная от

записи). MS-DOS обнаруживает и анализирует большинство таких ошибок и обеспечивает возможности для восстановления.

Высокий уровень

Интерпретатор Бейсика обнаруживает многие ошибки, включая ошибки драйверов устройств. При их обнаружении возвращается код ошибки, и если не предусмотрена программа восстановления при ошибках, программа останавливается. Однако можно установить обработку ошибок, с тем чтобы, когда происходит критическая ошибка, Бейсик автоматически переходил на процедуру восстановления при сбоях, созданную вами. Процедура может проанализировать код и определить, в какой строке программы произошла ошибка. После того как это сделано, программа может принять меры по устранению ошибки либо с помощью пользователя, либо выполняя другую часть программы. По завершении процедуры программа может продолжить выполнение с любого, нужного вам места (лишь с некоторыми ограничениями). Код для тщательного анализа ошибочных ситуаций может существенно увеличить размер программы. Отметим, что компилятор Бейсика фирмы IBM, даже для минимальной проверки на ошибки, потребует дополнительно не менее 4 байт на каждую строку программы.

Чтобы разрешить обработку ошибок в Бейсике, поместите в начале программы строку ON ERROR GOSUB n, где n - это номер строки программы, в которой начинается процедура обработки ошибок. При возникновении критической ошибки управление будет передано на эту строку. В начале процедуры поместите ряд строк вида IF ERR = n THEN номерстроки, где n - номер ошибки, взятый из приложения к руководству по Бейсику, содержащему сообщения об ошибках. Номера строк в этих операторах соответствуют началу кода, обрабатывающего данную конкретную ошибку. Эти части могут быть в свою очередь разбиты на куски рядом операторов IF ERL = n THEN номерстроки. ERL возвращает номер строки, в которой произошла ошибка, позволяя процедуре восстановления точно определить ошибочное место.

После того как процедура восстановления завершила свою работу, надо использовать оператор RESUME для возврата управления в ту строку, где произошла ошибка. За этим оператором может следовать номер, в этом случае управление будет передано на строку с указанным номером. Однако имейте в виду, что нельзя использовать RESUME для перехода в точку программы, которая находится за пределами процедуры, в которой произошла ошибка. Если восстановление после ошибки невозможно, но необходимо, чтобы программа продолжила свою работу, напишите RESUME NEXT, и управление будет передано на строку, следующую за

той, в которой произошла ошибка. Вот общая структура процедуры восстановления в Бейсике:

```
100 ON ERROR GOSUB 5000              'разрешаем обработку ошибок
.
.
5000 IF ERR = 61 THEN 5100           'диск полон
5010 IF ERR = 71 THEN 5200           'диск не готов
.
.
5100 IF ERL = 2080 THEN 5120         'где произошла ошибка?
5110 BEEP: PRINT "Disk in drive B: is full": RESUME
5120 BEEP: PRINT "Disk in drive A: is full": RESUME
.
5200 BEEP: PRINT "A disk drive is not ready" 'сообщение об ошибке записи
5210 PRINT "Strike any key when corrected"
5220 IF INKEY$ = "" THEN 5220        'ожидаем нажатия клавиши
5230 RESUME ERL - 10                 'пытаемся повторить операцию
```

В Бейсике 3.0 введены инструкции ERDEV и ERDEV$. Обе они позволяют получить переменные только для чтения от прерывания 24H, обрабатывающего критические ошибки. Z% = ERDEV возвращает в Z% слово статуса, в котором старший байт содержит 13-15 биты атрибута заголовка устройства, а младший байт - код ошибки прерывания 24H. Z$ = ERDEV$ помещает в Z$ 8-байтовое имя устройства для символьных устройств и 2-байтовый указатель накопителя для блочных устройств.

Низкий уровень

Иногда драйверы устройств содержат такие серьезные ошибки, что программа просто не может выполняться пока они не будут исправлены. Когда такие ошибки происходят, то система вызывает обработчик критических ошибок. Он может вступать в действие как для стандартных устройств, так и для установленных драйверов. Пользователь наиболее часто сталкивается с ним, когда пытается произвести дисковую операцию с дисководом, у которого открыта дверца. В этом случае появляется сообщение: "Not ready error reading drive A - Abort, Retry, Ignore?"

Обработчик критических ошибок может быть переписан с целью улучшения обработки устройств, для которых вы создали устанавливаемые драйверы. Вектор прерывания 24H указывает на стандартную процедуру MS-DOS, но вы можете перенаправить вектор на свою процедуру. При вызове этой процедуры старший бит AH содержит 0, если ошибка произошла на блочном уст-

ройстве, и 1, если на символьном. BP:SI указывают на заголовок драйвера виновного устройства, который может дать дополнительную информацию. Восемь байтов заголовка, начиная со смещения AH, содержат имя устройства, а обработчик критических ошибок помещает код ошибки длиной в слово в DI. Вот кодовые номера (они не представляют битовых позиций):

Код	Проблема
0	попытка писать на диск, защищенный от записи
1	неизвестное устройство
2	накопитель не готов
3	неизвестная команда
4	ошибка обмена данными
5	неверная длина запроса
6	ошибка поиска
7	неизвестный тип носителя
8	сектор не найден
9	нет бумаги в принтере
A	ошибка при записи
B	ошибка при чтении
C	общая ошибка

В случае дисковой ошибки AL содержит номер накопителя, на котором произошла ошибка (0 = A, 1 = B и т.д.), а биты 2-0 AH индицируют тип ошибки. Бит 0 устанавливается, если ошибка произошла во время операции записи, и сбрасывается, если при чтении. Биты 2-1 содержат информацию о том, в каком месте диска произошла ошибка, давая 00 для начальных секторов DOS, 01 для FAT, 10 для каталога и 11 для всего остального диска.

Имеется три способа, с помощью которых программа может восстановиться после критической ошибки.

1. Можно попросить пользователя устранить причину ошибки (например, закрыть дверцу накопителя), после чего система предоставит устройству возможность повторить операцию.
2. Управление может быть возвращено инструкции, следующей за INT 21H, которая сделала попытку обратиться к драйверу.
3. Программа может завершиться и вернуть управление системе.

Ваша процедура обработки ошибок может восстановить ситуацию, выдав инструкцию IRET, после того как она поместила 0 в AL, чтобы игнорировать ошибку, 1, чтобы повторить операцию, и 2, чтобы завершить программу. Если вы хотите, чтобы ваша про-

цедура провела восстановление сама, то она должна восстановить регистры выполняемой программы из стека, а затем удалить со стека все, кроме последних трех слов. После этого инструкция IRET возвратит управление программе, хотя сама система останется в нестабильном состоянии до тех пор, пока она не сделает вызов функции с номером, большим чем 12. Вот конфигурация стека (начиная сверху вниз) при вызове обработчика критических ошибок:

Адрес возврата обработчика ошибок:	IP, CS, флаги
Пользовательские регистры задачи, из которой был вызван драйвер:	AX, BX, CX, DX, SI, DI, BP, DS, ES, IP, CS, флаги

MS-DOS обрабатывает также многие некритические ошибки. Сюда включаются коды ошибок, которые могут возвращаться в регистрах, при обращении к функции DOS. Эти коды обсуждаются в данной книге в тех местах, в которых описываются соответствующие функции. Однако имейте в виду, что, начиная с версии 3.0, MS-DOS возвращает расширенные коды ошибок для функций, использующих FCB или дескрипторы файлов. Когда при выполнении одной из этих функций устанавливается флаг переноса, в AX возвращается обычный код ошибки. Дополнительный расширенный код доступен через прерывание 59H, если в BX поместить 0. Эта функция сообщает также о критических ошибках, и она может быть использована в обработчике критических ошибок, вызываемом через прерывание 24H.

Функция помещает в AX код ошибки, взятый из обычного списка знакомых кодов ошибок (например, "недостаточно памяти") или один из новых кодов (например, "ограничение доступа" для многопользовательской системы). BH возвращает код класса ошибки, указывая, какого типа произошла ошибка. Например, код 1 предупреждает, что исчерпаны ресурсы, т.е. что память, файловые буфера или что-то еще израсходовано. Другие классы могут указывать на программные ошибки, проблемы с носителями, форматированием и т.д. BL содержит код, предполагающий действие для восстановления, такие, как "повторить", "прекратить" или "запросить у пользователя". Наконец, CH возвращает число, определяющее место, где возникли проблемы: на блочном устройстве, на символьном, в памяти?

Данные для этих кодов ошибок весьма обширны. Полную информацию о них см. в Техническом руководстве по MS-DOS 3.0. Поскольку предполагается, что MS-DOS 3.0 не будет функционировать на машинах, более ранних, чем AT, то использование этих кодов ограничивает совместимость ваших программ. Тем не менее

набор процедур, предназначенный только для MS-DOS 3.0, может дополняться сверх обычных процедур обработки ошибок. В [1.1.3] показано, как программа может определить версию MS-DOS, в которой она работает.

Наконец, имейте в виду, что процесс может передавать код завершения вызвавшему его процессу. Термин *процесс* относится к взаимодействующим программам. Например, когда одна программа загружает и запускает другую с помощью функции EXEC, то запускаемая программа называется потомком, а запускающая - родителем. Родителю может потребоваться информация о том, как завершился потомок. Чтобы использовать эту возможность, поместите желаемый код завершения в AL и выполните функцию 4CH прерывания 21H для завершения программы. Когда управление будет возвращено родителю, то он выполнит функцию 4DH прерывания 21H (без входных регистров), и в AL будет получен код завершения, который может затем быть проанализирован. Кроме того, AH будет содержать информацию о том, как завершился потомок: 0 - для нормального завершения, 1 - по Ctrl-Break, 2 - по критической ошибке устройства и 3 - с помощью функции 31H, оставляющей программу резидентной.

Если программа завершилась с помощью этой функции (а не 20H, см. обсуждение в [1.3.4]), то MS-DOS получает код выхода и он может быть включен в обработку командным файлом с помощью подкоманды IF. Эта подкоманда позволяет условное исключение других команд из командного файла. Код выхода рассматривается как номер ERRORLEVEL и условные операции выполняются в зависимости от того, больше он определенного числа или нет. С помощью этой возможности командные файлы могут прекращать обработку и выводить сообщение о возникновении ошибки в одной из запущенных программ. Более подробная информация приведена в разделе "Команды пакетной обработки" руководства по операционной системе.

7.3. Использование специальных устройств ввода/вывода

Имеется огромное количество устройств ввода/вывода, которые могут быть присоединены к IBM PC, включая мышь, джойстик, графопостроители и т.д. В данном разделе обсуждаются только те устройства, которые специально поддерживаются оборудованием IBM PC, например кассетные магнитофоны,

световое перо и другие устройства, которые могут быть присоединены через игровой порт. Адреса портов, относящиеся к другим устройствам, обсуждаются в соответствующих разделах этой книги. Распределение адресов портов в основном одинаково для всех типов IBM PC:

Адрес порта	Функция
00-0F	микросхема DMA 8237 (не для PCjr)
20-2F	микросхема прерываний 8259 (AT контроллер #1: 20-3F)
40-4F	микросхема таймера 8253/8254
60-6F	микросхема PPI 8255 (AT использует только адреса клавиатуры)
70-7F	часы реального времени (только для AT)
80-83	регистры страниц DMA (не для PCjr)
A0-BF	микросхема прерываний #2 (только для AT)
C0-C7	микросхема звука SN76496 (только для PCjr)
F0-FF	PCjr - контроллер НГМД, AT - управление математическим сопроцессором
1F0-1F8	фиксированный диск AT
200-20F	игровой адаптер
278-27F	AT коммуникационный порт #2
2F8-2FF	коммуникационный порт COM2 (COM1 для PCjr)
320-32F	фиксированный диск XT
378-37F	адаптер параллельного принтера для PC, XT, AT
3B0-3BF	монохромный/параллельный адаптеры (не для PCjr)
3D0-3DF	цветной графический адаптер
3F0-3F7	контроллер НГМД
3F8-3FF	коммуникационный адаптер COM1 (модем PCjr)

7.3.1. Чтение/запись с кассетного магнитофона

Только очень немногие IBM PC и PCjr имеют кассетный магнитофон, а XT и AT не поддерживают его вообще. Помимо того, что он очень ненадежен, обмен с кассетным магнитофоном возможен только последовательный, но не с прямым доступом. Тем не менее могут быть причины для программирования кассетного магнитофона на PCjr. Имейте в виду, что кассетные операции используют канал 2 микросхемы таймера 8253 ([2.1.1]), поэтому не пытайтесь одновременно использовать этот канал для других целей. Отметим также, что при операции чтения с кассеты запре-

щено прерывание времени суток, поэтому счетчик времени суток BIOS будет давать неверное значение.

Высокий уровень

Хотя кассетные файлы обрабатываются совсем по-другому, чем дисковые, однако команды доступа к ним совершенно аналогичны. На кассету могут записываться только программные файлы и последовательные файлы данных. Последние могут включать файлы изображения памяти. Отметим, что данные не могут добавляться к последовательным файлам. При создании именам файлов даются следующие однобайтовые расширения:

.B	программа на Бейсике
.P	защищенная программа на Бейсике
.A	программа на Бейсике в формате ASCII
.M	файл изображения памяти
.D	последовательный файл данных

Для сохранения файла на кассете напишите SAVE "CAS1:имяфайла", для загрузки программы - LOAD "CAS1:имяфайла". В последнем случае лента прогоняется до тех пор, пока нужный файл не будет найден, при этом имя каждого встреченного файла будет выводиться на экран (кассеты не используют каталоги). Если запросить несуществующий файл, то будет выведен полный список файлов на кассете.

Средний уровень

BIOS работает с кассетной лентой порциями, состоящими из 256-байтовых блоков. Набору блоков предшествует "лидер", который состоит из 256 байт ASCII 1. Лидер завершается нулевым битом синхронизации. Затем следует байт синхронизации со значением 16H, а затем - 256 байт данных. После этого идут 2 байта контроля ошибок, а далее - новый блок данных, сопровождающийся парой байтов проверки ошибок и т.д. Вся последовательность завершается четырехбайтовым "хвостом", содержащим коды ASCII 1.

Для чтения данных с кассеты надо использовать функцию 2 прерывания 15H. Нет необходимости открывать файл, как это делается при дисковых операциях. ES:BX указывают на буфер в памяти, куда будут посылаться данные, а CX - число байтов, которые надо считать. При возврате DX сообщит, сколько байтов прочитано на самом деле, а ES:BX будут указывать на последний счи-

танный байт плюс 1. Флаг переноса будет равен 0, если чтение прошло успешно, в противном случае AH будет содержать 1, если проблема с контролем ошибки, 2 - при ошибке чтения данных и 3 - при отсутствии данных на ленте.

Функция 3 прерывания 15H записывает данные на кассету. ES:BX указывают на первый байт данных, а CX содержит число байтов, которое надо записать. При возврате ES:BX указывают на байт, следующий за последним записанным. Мотор управляется функциями 0 (включение) и 1 (выключение) прерывания 15H. Для этих функций нет выходных регистров.

7.3.2. Чтение позиции светового пера

Хотя компьютеров, оснащенных световым пером довольно мало, тем не менее это одно из немногих вспомогательных устройств, которое поддерживается как оборудованием, так и операционной системой. Световое перо работает с помощью небольшого оптического детектора, находящегося на его кончике. По ходу сканирования экрана электронным лучом инициируется импульс оптического детектора, когда пучок достигает точки экрана, над которой находится перо. Время возникновения этого импульса относительно сигналов горизонтальной и вертикальной синхронизации позволяет определить позицию светового пера.

Высокий уровень

Бейсик может определять позицию светового пера двумя способами. При первом программа непрерывно определяет статус пера. При втором управление временно (до окончания работы пера) передается процедуре, обеспечиваемой вашей программой. Для непрерывного контроля за пером надо использовать оператор PEN как функцию в форме X = PEN(n), где n - кодовый номер, определяющий, какую информацию о пере и его позиции вы хотите получить. Возможные значения n таковы:

0 возвращает -1, если перо было выключено со времени последнего запроса, 0 - если нет

1 возвращает последнюю координату x (0-319 или 0-639), в которой перо было включено (оно могло быть впоследствии передвинуто, если оставалось включенным)

2 возвращает последнюю координату y (0-199), в которой перо было включено.

3 возвращает -1, если перо включено, и 0 - если нет

4 возвращает текущую x координату (0-319 или 0-639) пера

5 возвращает текущую y координату (0-199) пера

6 возвращает позицию - номер строки (1-24), в которой перо было последний раз активизировано

7 возвращает позицию - номер столбца (1-40 или 1-80), в которой перо было последний раз активизировано

8 возвращает текущую позицию - номер строки (1-24)

9 возвращает текущую позицию - номер столбца (1-40 или 1-80)

В данном примере определяется, включено ли перо, и если это так, то устанавливается его текущее положение:

```
100 IF NOT PEN(3) THEN 130      'проверяем, включено ли перо
110 X = PEN(4)                  'получаем координату точки по оси x
120 Y = PEN(5)                  'получаем координату точки по оси y
130 ...                         'продолжаем программу
```

Более гибкие возможности использования светового пера предоставляются оператором ON PEN GOSUB. Этот оператор указывает номер строки, в которой начинается процедура, активизируемая при включении пера. Бейсик достигает этого проверкой состояния пера после выполнения каждой его инструкции. Процедура может определить позицию пера и предпринять требуемые действия. Когда процедура заканчивается, программа продолжается с того места, где она была при включении пера.

ON PEN GOSUB не работает до тех пор, пока она не активизирована оператором PEN ON. PEN OFF отменяет ее работу. Смысл этого в том, что постоянная проверка статуса пера замедляет работу программы, поэтому ее надо осуществлять, только когда это необходимо. Если программа начинает выполнять критическую часть кода, когда нельзя использовать процедуру ON PEN GOSUB, напишите PEN STOP. В этом случае будет продолжаться проверка статуса пера, и если перо будет включено, то этот факт будет запомнен. Однако пока не будет встречен оператор PEN ON, управление не будет передаваться процедуре ON PEN GOSUB.

В следующем примере остановка программы происходит в момент нажатия кнопки на световом пере. Точка в позиции светового пера включается процедурой, обрабатывающей включение светового пера.

```
100 ON PEN GOSUB 5000           'устанавливаем процедуру для светового
110 PEN ON                      'пера и включаем режим его отслеживания
```

```
5000 '''процедура обработки светового пера
5010 X = PEN(4)              'получаем координату X
5020 Y = PEN(5)              'получаем координату Y
5030 PSET(X,Y)               'включить эту точку
5040 RETURN
```

Средний уровень

Функция 4 прерывания 10H BIOS сообщает текущую позицию светового пера. У нее нет входных регистров. При возврате AX содержит 0, если перо не включено, и 1, если получены новые значения для его позиции. Возвращаются два набора координат: позиции точки и позиции строки и столбца. Позиции символа содержатся в DX, причем DH содержит строку (0-24), а DL - столбец (0-79). Позиция точки содержится в CH и BX, где CH содержит вертикальную координату (0-199), а BX - горизонтальную (0-319 или 0-639 в зависимости от режима терминала).

```
;---читаем и запоминаем положение светового пера
MOV  AH,4              ;номер функции
INT  10H               ;прерывание BIOS
CMP  AH,1              ;новая позиция?
JE   NO_READING        ;если нет, то уходим
MOV  COL,BX            ;сохраняем горизонтальную координату
MOV  CL,CH             ;помещаем вертикальную координату
MOV  CH,0              ;в CX
MOV  ROW,CX            ;сохраняем вертикальную координату
```

Низкий уровень

По своей сути световое перо является расширением видеосистемы и как таковое использует микросхему контроллера CRT 6845. Позиция светового пера дается одним двухбайтовым значением, хранящимся в регистрах 10H (старший байт) и 11H (младший байт) микросхемы. В [4.1.1] объясняется, как читать регистры микросхемы. Посмотрите пример. Порт с адресом 3DCH устанавливает задвижку светового пера, а с номером 3DBH сбрасывает ее.

```
;---проверка светового пера и чтение его позиции
MOV  DX,3DAH           ;указываем на регистр статуса
IN   AL,DX             ;получаем информацию
TEST AL,4              ;проверяем выключатель
JNZ  NOT_SET           ;на выход
```

```
TEST AL,2          ;проверяем триггер
JZ   NOT_SET       ;на выход
SUB  DX,7          ;указываем на регистр адреса 6845
MOV  AL,10H        ;запрос на старший байт позиции пера
OUT  DX,AL         ;посылаем запрос
INC  DX            ;указываем на регистр данных 6845
IN   AL,DX         ;получаем значение
XCNG AH,AL         ;запоминаем его в AH
DEC  DX            ;возвращаемся к адресному регистру
MOV  AL,11H        ;теперь хотим получить младший байт
OUT  DX,AL         ;посылаем запрос
INC  DX            ;назад к регистру данных
IN   AL,DX         ;теперь это значение в AX
```

7.3.3. Получение аналогового ввода через игровой порт

Игровой порт может поддерживать 2 джойстика или 4 "ручки"*. Для джойстика он сообщает две координаты и статус двух кнопок; для ручки он сообщает одну координату и статус одной кнопки. Несколько вспомогательных устройств, таких, как графическое табло, также может быть подключено к игровому порту; их работа может осуществляться параллельно с работой джойстика. Данный раздел посвящен чтению координат, а в следующем обсуждается, как получить статус кнопок.

Высокий уровень

Функция STICK возвращает позиции по осям, указываемые следующими кодовыми номерами:

0 ось X джойстика A
1 ось Y джойстика A
2 ось X джойстика B
3 ось Y джойстика B

Вам нужно написать, например, X = STICK(0) и в X будет содержаться значение координаты X для джойстика A. Но эта функция имеет одну особенность, о которой вам необходимо знать.

---

*Здесь и далее термином "ручка" мы будем называть специальную ручку для управления игрой (англ. paddle). В Советском Союзе это устройство мало распространено, и мне неизвестно общеупотребительное его название. - *Примеч. пер.*

Только при использовании кода 0 действительно читаются координаты джойстика, при этом читаются все 4 значения. Кодовые номера 1-3 просто сообщают показания, прочитанные кодом 0. Чтобы получить последние 3 координаты, вам необходимо перед этим использовать функцию X = STICK(0), даже если вам не нужно знать значение, возвращаемое кодом 0.

Джойстики различаются по своим физическим характеристикам, поэтому необходимо настраивать их, чтобы их предельные положения совпадали с границами экрана. В следующем примере показано, как это делается. В примере непрерывно рисуется точка в позиции, указываемой джойстиком, действие, которое требуется, чтобы диапазон значений, принимаемых игровым портом, преобразовывался в диапазон позиций экрана.

```
100 '''получаем предельные показания джойстика
110 STRIG ON                                    'разрешаем кнопки
120 V = STRIG(0)                                'чистим старые показания
130 PRINT "Briefly push button 1 when stick is farthest to left"
140 XLEFT = STICK(0)                            'получаем самое левое значение
150 IF STRIG(0) = 0 THEN 140                    'ждем нажатия кнопки
160 STRIG OFF: FOR N = 1 TO 1000: NEXT: STRIG ON
170 PRINT "Briefly push button 1 when stick is farthest to right"
180 XRIGHT = STICK(0)                           'получаем самое правое значение
190 IF STRIG(0) = 0 THEN 180                    'ждем нажатия кнопки
200 STRIG OFF: FOR N = 1 TO 1000: NEXT: STRIG ON
210 PRINT "Briefly push button 1 when stick is farthest to top"
220 V = STICK(0): YTOP = STICK(1)               'самое верхнее значение
230 IF STRIG(0) = 0 THEN 220                    'ждем нажатия кнопки
240 STRIG OFF: FOR N = 1 TO 1000: NEXT: STRIG ON
250 PRINT "Briefly push button 1 when stick farthest to bottom"
260 V = STICK(0): YBOT = STICK(1)               'самое нижнее значение
270 IF STRIG(0) = 0 THEN 260                    'ждем нажатия кнопки
280 STRIG OFF                                   'закончили
290 '''получаем множители для установки размера экрана
300 XRIGHT = XRIGHT - XLEFT                     'горизонтальный размер
310 XMULTIPLIER = 320/XRIGHT                    'вычисляем число точек на деление
320 YBOT = YBOT - YTOP                          'вертикальный размер
330 YMULTIPLIER = 200/B/YBOT                    'число точек на деление
340 '''теперь вычисляем координаты в режиме умеренного разрешения
350 X = STICK(0)                                'получаем горизонтальную позицию
360 Y = STICK(1)                                'получаем вертикальную позицию
```

```
370 X = (X - XLEFT)*XMULTIPRIER    'приводим к экрану
380 Y = (Y - YTOP)*YMULTIPRIER
390 PSET(X,Y)                      'выводим точку в нужной позиции
400 GOTO 350                       'повторяем
```

Средний уровень

Только AT предоставляет поддержку джойстика на уровне операционной системы. Это функция 84H прерывания 15H, которая возвращает координаты, причем:

```
AX =    координата X джойстика A
BX =    координата Y джойстика A
CX =    координата X джойстика B
DX =    координата Y джойстика B
```

При входе поместите в DX 1. Когда в DX содержится 0, функция возвращает состояние кнопок джойстика [7.3.4]. Когда у машины нет игрового порта, при возврате устанавливается флаг переноса.

Низкий уровень

Информация о координатах обоих джойстиков или всех четырех ручек хранится в одном байте, доступ к которому осуществляется через порт 201H. Вот значение его битов:

Бит	Джойстик	Ручка
3	координата Y джойстика B	координата ручки D
2	координата X джойстика B	координата ручки C
1	координата Y джойстика A	координата ручки B
0	координата X джойстика A	координата ручки A

Координата может описываться одним битом за счет измерения временных интервалов. Пошлите в порт байт с произвольным значением. Это приведет к тому, что младшие 4 бита обнулятся. Затем постоянно считывайте значение из порта, замеряя интервал времени, через который интересующий вас бит станет равным 1. Этот интервал пропорционален позиции джойстика по интересующей вас оси. Максимальные интервалы соответствуют нижней позиции по оси Y и правой позиции по оси X. Независимо от

позиции, биты меняются от 0 к 1 очень быстро, по сравнению с механической скоростью перемещения джойстика или ручки. Поэтому программа может с большой точностью получить сначала позицию координаты Y, а затем позицию координаты X. Нет необходимости тестировать каждую координату отдельно. В предлагаемом ниже примере определяется координата X джойстика A.

```
;---получаем координату X джойстика A
        MOV  DX,201H     ;адрес игрового порта
        OUT  DX,AL       ;посылаем в порт произвольное значение
        MOV  AH,1        ;проверяем бит 1
        MOV  SI,0        ;инициализируем счетчик
NEXT:   IN   AL,DX       ;читаем значение из порта
        TEST AL,AH       ;проверяем бит 1
        JE   FINISHED    ;выход, когда бит установлен
        INC  SI          ;иначе, увеличиваем счетчик
        LOOP NEXT        ;на начало цикла
FINISHED:                ;теперь в SI находится X-координата ручки A
```

7.3.4. Получение цифрового ввода из игрового порта

Игровой порт поддерживает два джойстика или четыре ручки, а также ряд графических устройств. При этом одновременно может определяться статус не более четырех кнопок устройств. Проверка состояния кнопок оказывается сложным делом, поскольку программа не имеет возможности постоянно проверять их, а кнопка может быть нажата и отпущена, пока программа занята другими делами. Для решения этой проблемы создают специальную процедуру. Статус кнопок автоматически читается несколько раз в секунду без специального запроса на это программы; когда оказывается, что кнопка нажата, то управление передается процедуре, которая определяет, какая кнопка нажата и предпринимает нужные действия.

Высокий уровень

Бейсик использует оператор STRIG для чтения статуса кнопок. Оператор STRIG достаточно "хитрый", чтобы обнаружить нажатие кнопки, даже если программа в данный момент не проверяет статус кнопки, т.е. программа может запросить: "Было ли нажатие кнопки, со времени моего последнего запроса?". Это свойство очень полезно для видеоигр, поскольку программа может зани-

маться манипуляциями с экраном, не заботясь о постоянной проверке статуса кнопок. Однако это существенно замедляет скорость выполнения программы, поскольку проверка статуса кнопок осуществляется после выполнения каждого оператора. По этой причине STRIG работает, только когда он преднамеренно включен, а он может включаться и выключаться по ходу программы.

STRIG работает двумя способами. Во-первых, он может работать как функция, которая непосредственно читает текущие значения кнопок, в форме X = STRIG(n). Здесь n - кодовый номер:

0 кнопка A1 нажата со времени последнего вызова
1 кнопка A1 в данный момент отпущена
2 кнопка B1 нажата со времени последнего вызова
3 кнопка B1 в данный момент отпущена
4 кнопка A2 нажата со времени последнего вызова
5 кнопка A2 в данный момент отпущена
6 кнопка B2 нажата со времени последнего вызова
7 кнопка B2 в данный момент отпущена

Во всех случаях функция возвращает -1, если описание применимо, и 0, если нет.

Второй способ применения STRIG это форма, в которой он автоматически передает управление процедуре при нажатии кнопки. Надо написать ON STRIG(n) GOSUB номерстроки. Номер строки дает начальный номер строки процедуры. Число n относится к кнопке, где 0 = A1, 2 = B1, 4 = A2 и 6 = B2. Каждая кнопка может обрабатываться своей процедурой или может быть одна процедура для всех кнопок.

Для активизации функции STRIG включите в программу оператор STRIG(n) ON. В качестве значения n используются четыре перечисленных кода. Чтобы отменить его (ускоряя работу программы), напишите STRIG(n) OFF. Имеется также третья возможность. STRING(n) STOP приводит к тому, что нажатие кнопки запоминается, но никаких действий не предпринимается до очередного оператора STRING(n) ON. Это свойство предохраняет от нежелательных перерывов из-за оператора ON STRING GOSUB. Однако при выполнении условия STRIG(n) STOP программа замедляется.

В следующем примере показано действие ON STRIG GOSUB. В примере параграфа [7.3.3] содержатся строки, демонстрирующие форму X = STRIG.

```
100 ON STRIG(0) GOSUB 5000'переход на 5000 при нажатии кнопки A1

200 STRIG(0) ON              'включаем проверку нажатия кнопки
```

```
300 STRIG(0) STOP          'отменяем уход на процедуру

400 STRIG(0) ON            'возобновляем уход на процедуру

500 STRIG(0) OFF           'отменяем проверку нажатия кнопки

5000 '''здесь находится процедура обработки нажатия кнопки A1

5500 RETURN                'возврат к тому месту, откуда попали сюда
```

Средний уровень

Только AT предоставляет поддержку джойстика на уровне операционной системы. Функция 84H прерывания 15H возвращает установку кнопок в битах 4-7 регистра AL, как показано ниже. При входе DX должен содержать 0; когда DX содержит 1, то тогда возвращаются координаты джойстика (см. [7.3.3]). Если машина не имеет игрового порта, то при возврате устанавливается бит переноса.

```
;---проверяем кнопку #2 джойстика B (бит 7)
MOV  AH,84H          ;номер функции
MOV  DX,0            ;запрос состояния кнопок
INT  15H             ;вызов функции
JC   NO_JOYSTICK     ;если нет джойстика, то на выход
TEST AL,10000000B    ;проверяем бит 7
JNZ  BUTTON_DOWN     ;переход, если кнопка нажата
```

Низкий уровень

Биты 7-4 порта с адресом 201H содержат статус кнопок, связанных с игровым портом. Значение битов меняется в зависимости от того, присоединены\ли джойстики или ручки:

Бит	Джойстик	Ручка
7	Кнопка #2 джойстика B	Кнопка ручки D
6	Кнопка #1 джойстика B	Кнопка ручки C
5	Кнопка #2 джойстика A	Кнопка ручки B
4	Кнопка #1 джойстика A	Кнопка ручки A

Программе нужно просто прочитать значение из порта и проверить установку соответствующих битов:

```
MOV  DX,201H        ;адрес порта игрового адаптера
IN   AL,DX          ;получаем значение из него
TEST AL,0010B       ;проверяем бит 1 (кнопка A2 нажата?)
JNZ  BUTTON_A2      ;если да, то на процедуру обработки
```

Программа имеет обычно более важные дела, чем постоянная проверка игрового порта, однако настолько же бессмысленной оказывается периодическая проверка порта путем помещения процедуры в разные части программы. Чтобы получить эффект слежения за нажатием кнопок, аналогичный описанному в Бейсике, вам придется создать дополнение к прерыванию времени суток, как описано в [2.1.7]. Прерывание обычно выполняется 18,2 раза в секунду, и каждый раз вы можете проверять игровой порт и при необходимости предпринимать нужные действия.

Приложение А. Двоичные и шестнадцатиричные числа и адресация памяти

Основной единицей хранения данных в компьютере является бит. В большинстве микрокомпьютеров восемь битов объединены в байт, при этом каждый бит байта может быть установлен ("включен" (= 1)) или сброшен или ("выключен" (= 0)), допуская 256 разных вариантов. Таким образом, в одном байте можно представить 256 различных символов (расширенный набор кодов ASCII) или целое число в диапазоне от 0 до 255. Хотя мы привыкли записывать эти числа в десятичной форме, они могут записываться также в двоичной или шестнадцатиричной форме - их значения при этом не изменяются, а программы могут с одинаковой легкостью читать эти значения как в той, так и в другой форме. Вместо того чтобы говорить, что в одном байте могут храниться числа от 0 до 255, можно сказать, что могут храниться двоичные числа от 00000000 до 11111111 или шестнадцатиричные числа от 00 до FF. Поскольку можно перепутать разные формы, то двоичные и шестнадцатиричные числа отмечаются специальным образом. В языке ассемблера за двоичными числами следует буква B, а за шестнадцатиричными числами - буква H, например, 11111111B или FFH. Бейсик фирмы Microsoft предваряет шестнадцатиричные числа символами &H, например &FFH; к сожалению, числа в двоичной форме он не распознает.

Двоичные числа

Когда содержимое байта представляется в двоичной форме, то требуется 8 цифр. Каждая цифра соответствует одному биту; биты нумеруются от 0 до 7. Как и в десятичных числах, цифры располагаются справа налево, от младших к старшим разрядам. В отличие от десятичных чисел, в которых каждая последующая цифра в 10 раз больше своей соседки справа, двоичные цифры увеличиваются только в два раза. Таким образом, самая правая цифра считает единицы, вторая - двойки, третья - четверки и т.д. до значения 128 для восьмой цифры байта. Это означает, что если первая цифра равна 1, то прибавление к ней 1 приводит к тому, что она

станет 0, а 1 будет перенесена во/вторую цифру, как для десятичных чисел 9 + 1 = 0 и перенос единицы в следующий разряд. Вот как числа первого десятка представляются в двоичной форме:

00000000	0
00000001	1
00000010	2
00000011	3
00000100	4
00000101	5
00000110	6
00000111	7
00001000	8
00001001	9
00001010	10

В этой последовательности большинство нулей слева необязательно, т.е. ее можно записать и в виде 0, 1, 10, 11, 100, 101 и т.д. Нули включены только для того, чтобы напомнить вам, что байт состоит из восьми цифр, соответствующих битам. Возможно, вам покажется, что работать с набором нулей и единиц несколько утомительно, поэтому вы можете облегчить задачу, если будете представлять себе двоичные числа следующим образом:

бит	7	6	5	4	3	2	1	0
значение	128	64	32	16	8	4	2	1

Когда вы встречаете двоичное число 10000001, то установлены биты 7 и 0. Бит 7 соответствует 128, а бит 0 - 1, поэтому десятичное значение байта равно 129. Если этот байт представляет символ, то он соответствует коду ASCII 129, который представляет букву u с умляутом (в альтернативной кодировке ГОСТа - букву Б). Наоборот, чтобы определить цепочку битов для буквы A, равной ASCII 65, просмотрите приведенную выше таблицу на значения битов, которые она содержит: 64 и 1, что соответствует 01000001B.

Зачем нужно связываться с двоичными числами? Одна из причин состоит в том, что компьютер хранит информацию в статусных байтах памяти и статусных регистрах микросхем. Отдельные части этой информации распределены по одному или двум байтам. Это достигается назначением определенных битов определенным данным. Например, помимо других вещей, байт статуса может

сообщить, сколько принтеров и дисковых накопителей присоединено к вашей машине. Скажем, два старших байта содержат число принтеров, а два младших - число дисковых накопителей. Байт статуса расположен в определенной ячейке памяти и, как и любой байт, может иметь значения от 0 до 255. Если значение этого байта равно 66 или 01000010 в двоичной форме, то два старших байта равны 01, а два младших байта - 10. Первая пара говорит о том, что у нас имеется один принтер, а вторая - что 2 дисковых накопителя. Группа битов, рассматриваемая совместно таким образом, называется *полем*. Часто вашим программам приходится читать статусные байты или регистры, а иногда изменять установку битов. Для языка ассемблера эти операции тривиальны, но не для Бейсика. В приложении Б объясняется, как они выполняются в Бейсике.

Шестнадцатиричные числа

В то время как в двоичных числах каждая последующая цифра вдвое больше предыдущей, в шестнадцатиричных числах каждая последующая цифра больше в 16 раз. У десятичных чисел первая позиция соответствует единицам, вторая - десяткам, третья - сотням. У двоичных чисел первая позиция соответствует единицам, вторая двойкам, третья - четверкам. У шестнадцатиричных чисел первая позиция соответствует единицам, вторая - 16, третья - 256 и т.д. Это означает, что когда в позиции единиц расположена цифра 9, то прибавление единицы не приводит к переносу в следующий разряд, как это было бы в случае десятичных чисел. Но как записать десятичное число 10 одной цифрой? Ответ состоит в том, что шестнадцатиричные числа используют первые 6 букв латинского алфавита в качестве дополнительных цифровых символов:

Шестнадцатиричный символ	Десятичный эквивалент
A	10
B	11
C	12
D	13
E	14
F	15

Перечисление шестнадцатиричных чисел продолжается так: ... 8, 9, A, B, C, D, E, F, 10, 11 ... 19, 1A, 1B и т.д.

Полезность шестнадцатиричных чисел опирается на тот факт, что одна шестнадцатиричная цифра описывает содержимое ровно 1/2 байта. Например, в числе F6 F соответствует старшим четырем битам байта, а 6 - младшим четырем битам (четыре бита, взятые вместе, называются ниблом, или по-русски "огрызком"). Несложно вычислить двоичный эквивалент четверки битов. FH = = 1111B, а 6H = 0110B (напоминаем, что H и B - суффиксы, помогающие отличить 11 двоичное от 11 десятичного и 11 шестнадцатиричного). Таким образом, число F6H представляет цепочку битов 11110110. Двухбайтовое число (целое) может равняться 6FF6H. В этом случае цепочка битов для него имеет вид 0110111111110110. Если число состоит только из трех цифр, то верхняя половина старшего байта равна нулю, например, числу F6FH соответствует цепочка битов 0000111101101111.

Шестнадцатиричные числа намного легче читать, чем двоичные. А после небольшой практики оказывается, что работать с ними намного удобнее, чем с десятичными.

Адреса памяти и портов

Теперь, когда вы научились работать с шестнадцатиричными числами, можно разбираться в системе, которой пользуется процессор при адресации памяти. Во-первых, важно отметить, что имеются два типа адресов: адреса памяти и адреса портов. Номера адресов, используемые теми и другими, совершенно не связаны; засылка значения в ячейку памяти с адресом 2000 не имеет ничего общего с засылкой значения в порт с адресом 2000. Доступ к портам осуществляется с помощью инструкций INP и OUT в Бейсике и IN и OUT в языке ассемблера. Доступ к адресам памяти осуществляется в Бейсике инструкциями PEEK и POKE, а в языке ассемблера - инструкцией MOV. Имеется 65K доступных адресов портов и 1024K доступных адресов памяти.

Поскольку процессор использует 16-битовые регистры, то он намного быстрее вычисляет адреса памяти, если они не превосходят по длине 16 битов. Однако максимальное число, которое может содержаться в 16 битах, равно 65535. Мы будем представлять его как четырехзначное шестнадцатиричное число FFFFH. Требуется еще 4 добавочных бита, чтобы представить такое большое число, как миллион (FFFFH), которому равен размер адресного пространства IBM PC (AT может иметь доступ к еще большей памяти, используя виртуальную адресацию, здесь не рассматриваемую).

Процессор решает проблему адресации более чем 64K с помощью 16-битового указателя за счет разбиения памяти на сегменты. Сегментом является любая непрерывная область памяти

размером 64K; при этом 16-битовый указатель может давать адрес любого байта внутри него. Процессор хранит положение начала сегмента в мегабайтовом адресном пространстве и рассматривает 16-битовые адреса как смещения относительно этой точки. Но как определить эту точку? Ответ состоит в том, что второе двухбайтовое значение используется для отметки начала сегмента и это значение умножается на 16 (= 4 битам) перед вычислением полного адреса. Таким образом, если это сегментное значение равно 2, то, умножив его на 16, получим 32, и адреса будут затем вычисляться как смещение относительно 32-го байта в памяти. Если адрес в сегменте равен 7, то, суммируя 32 и 7, получаем, что нам нужно обратиться к 39-му байту памяти, а не к 7-му. Относительный адрес (или смещение) этого байта равен 7, а абсолютный - 39.

В Бейсике вы можете установить сегментный адрес с помощью оператора DEF SEG. Если вы напишете оператор DEF SEG = 2, то установите начало сегмента, к которому будете обращаться на 32-й байт, как в предыдущем примере. Затем вы можете использовать операторы PEEK и POKE для чтения и записи отдельных байтов памяти. Например, PEEK(7) прочитает седьмой байт с начала сегмента, т.е. 39-й байт памяти.

Во многих местах этой книги мы ссылаемся на абсолютные адреса памяти. Это необходимо, поскольку операционная система хранит важную информацию в определенных местах. Абсолютные адреса приводятся в виде 0000:0000, где первые 4 шестнадцатиричные цифры указывают адрес сегмента, а вторые - относительный адрес (смещение). Вспоминая предыдущий пример, мы можем адресовать 39-й байт памяти, записав 0002:0007. Отметим, что тот же самый адрес может быть записан в другом виде, если изменить значение сегментного регистра, например 0001:0017. Этот адрес можно представить также в виде одного пятизначного шестнадцатиричного числа. Например, видеобуфер начинается с адреса B000:0000, который можно записать как B0000H. Отметим, что суффикс H опускается в специальной адресной нотации.

И последнее замечание относительно использования памяти. Когда число занимает два или более байта, то младший байт этого числа хранится в ячейке с меньшим адресом. Если целое число A48BH хранится, начиная с ячейки 1000:0007, то ячейка 0007 содержит 8B, а 0008 - A4. Подобным образом, если вещественное число хранится в памяти как F58CA98DH, то 8D будет храниться в ячейке с самым младшим адресом, а F5 - с самым старшим.

Приложение Б. Битовые операции в Бейсике

В Бейсике нельзя использовать числа в двоичной форме. Он рассматривает цепочку битов 11000000 как 11 миллионов, а не как 192. Однако манипуляции с цепочками битов часто необходимы при программировании, поскольку требуется читать и изменять содержимое статусных байтов и статусных регистров.

В большинстве случаев к цепочкам битов применяются две логические операции. Это операции ИЛИ (OR) и И (AND), и обе они доступны в Бейсике. Используемые по отдельности или в комбинации, они позволяют программе читать и устанавливать индивидуальные биты байта. Обе операции бинарные, т.е. они применяются к паре значений, давая в качестве результата третье, как обычные арифметические операции: Z = X OR Y. При использовании со значениями байтовой длины эти операции выполняются 8 раз, по разу для каждого бита. ИЛИ проверяет бит 0 двух байт, и если этот бит установлен хотя бы в одном байте, бит 0 будет установлен и в результирующем байте. Этот процесс выполняется и для остальных семи пар битов. Операция И устанавливает бит результата только в том случае, если оба бита были установлены, в противном случае этот бит будет равен 0. Изучите эти две операции, используя приведенную диаграмму:

	операнд1		операнд2		результат	операнд1		операнд2		результат
бит 7	1		0		1	1		0		0
6	1		0		1	1		0		0
5	1		1		1	1		1		1
4	1	OR	1	=	1	1	AND	1	=	1
3	0		0		0	0		0		0
2	0		0		0	0		0		0
1	0		1		1	0		1		0
0	0		1		1	0		1		0

При программировании ИЛИ применяется для установки одного или более битов в ячейке памяти или статусном регистре. Например, может возникнуть необходимость установить атрибут мерцания определенному символу на экране терминала. Эта операция требует установки седьмого бита. Программа может просто записать весь байт атрибутов по нужному адресу, но состояние

остальных семи битов может быть неизвестным. Поэтому надо прочитать байт из нужного места видеобуфера и поместить его в целую переменную, например X. Затем готовится байт, у которого установлен только седьмой бит. Как вы знаете (или можете узнать из приложения A), такой байт равен 128. Теперь просто запишите Y = X OR 128 и в Y будет то же значение, что и в X, но с установленным седьмым битом. Приведенная ниже диаграмма иллюстрирует эту операцию:

бит	атрибут		128		результат
7	0		1		1
6	1		0		1
5	0		0		0
4	1	OR	0	=	1
3	0		0		0
2	0		0		0
1	0		0		0
0	1		0		1

В числе 128 только седьмой бит установлен. Независимо от того, был ли он установлен или нет у байта атрибутов, он будет установлен в результирующем байте. Что касается остальных семи битов, то они будут установлены в результирующем байте, только если они уже были установлены в байте атрибутов. Операция ИЛИ может применяться для установки более чем одного бита за один прием (но обратите внимание на предостерегающие замечания, помещенные ниже). Чтобы установить биты 2 и 3, надо использовать сумму значений этих двух битов: 4 + 8 = 12.

бит	атрибут		12		результат
7	0		0		0
6	1		0		1
5	0		0		0
4	1	OR	0	=	1
3	0		1		1
2	0		1		1
1	0		0		0
0	1		0		1

Для сброса одного или более битов используется операция И. Для этого надо вычислить значение байта, у которого установлены все биты, за исключением того, который вы хотите сбросить. Помните, что все соответствующие биты должны быть установлены,

чтобы результирующий бит тоже был установлен. Чтобы сбросить бит 7, возьмите значение 255 - 128 = 127:

бит	атрибут		127		результат
7	1		0		0
6	1		1		1
5	0		1		0
4	1	AND	1	=	1
3	0		1		0
2	0		1		0
1	0		1		0
0	1		1		1

Отметим, что каждый бит, установленный в байте атрибутов (кроме бита 7), комбинируется с 1 в байтовом значении 127 и поэтому равен 1 и в результирующем байте. Биты, которые были равны 0 в байте атрибутов, остаются равными 0.

Иногда программе нужно установить группу битов (поле). Например, вы хотите изменить три младших бита байта видеоатрибутов, изменяя тем самым цвет символа. Пусть новая цепочка битов будет 101. Ей соответствует значение 5, но выполнение операции ИЛИ с 5 может не привести к желаемому результату, поскольку ИЛИ устанавливает бит в результирующем байте, если хотя бы один из соответствующих битов был равен 1. Если средний бит был установлен в байте атрибутов, то он останется установленным и в результирующем байте:

бит	атрибут		5		результат
2	0		1		1
1	1	OR	0	=	1
0	0		1		1

В таком случае программа должна сначала сбросить все три бита с помощью И, а затем установить нужные биты с помощью ИЛИ. В данном случае 255 - 4 - 2 - 1 = 248, поэтому сначала надо вычислить Y = X AND 248, а затем Z = Y OR 5.

Не слишком сложно для программы и определить, установлен или сброшен определенный бит. Для этого надо произвести операцию И с байтом, у которого сброшены все биты, кроме тестируемого (скажем, бита 5, который равен 32). Если результат отличен от нуля, то тестируемый бит был установлен:

бит	атрибут		32		результат
7	1		0		0
6	0		0		0
5	1		1		1
4	1	AND	0	=	0
3	0		0		0
2	0		0		0
1	0		0		0
0	1		0		0

Ну а что делать, когда программе требуется знать установку двух или более битов? Например, биты 6 и 7 могут хранить номер от 0 до 3, но если изолировать эти биты, то они дадут в результате одно из четырех (десятичных) значений: 0, 64, 128 и 192. Поскольку Бейсик вынуждает вас работать не с двоичными числами, то требуются особые приемы, чтобы определить, какому значению соответствует данная цепочка битов. Чтобы позволить вам избежать этих манипуляций, приводятся две процедуры. Первая из них преобразует десятичное число, хранящееся в байте, в строку из восьми 1 или 0. Отметим, что это символьная строка, а не двоичное число. Вторая процедура берет такую строку (любой длины) и преобразует ее в десятичное число. Используя эти процедуры, вы можете легко анализировать статусные байты в памяти. С помощью функции MID$ вы можете выделить битовое поле и преобразовать содержащееся в нем значение в десятичное число. Вот процедура преобразования десятичного числа в двоичную строку:

```
100 STATUSBYTE = PEEK(13): GOSUB 1000 'получаем байт из памяти и
                                      'выполняем процедуру преобра-
                                      'зования в двоичное число
110 PRINT BITPATTERN$                 'печатаем получившуюся строку
.
.
1000 '''преобразуем десятичное число в строку из 8 единиц и нулей
1010 BITPATTERN$ = ""                 'чистим переменную
1020 FOR N = 7 TO 0 STEP -1           'идем назад, начиная с бита 7
1030 IF STATUSBYTE - 2^N < 0 THEN 1060 'переход, если бит равен 0
1040 BITPATTERN$ = "1" + BITPATTERN$ 'добавляем к строке 1
1050 STATUSBYTE = STATUSBYTE - 2^N: GOTO 1070
1060 BITPATTERN$ = "0" + BITPATTERN$ 'добавляем к строке 0
```

```
1070 NEXT                              'повторяем 7 раз
1080 RETURN
```

Важно отметить, что порядок битов в двоичной строке обратный. Вместо того чтобы двигаться слева направо от бита 7 к биту 0, бит 0 расположен самым левым в строке. Причиной этого является тот факт, что функция MID$ может легко выделить требуемые вам биты. Поскольку MID$ начинает отсчет с 1, то вы должны считать, что биты пронумерованы от 1 до 8. Чтобы выделить четвертый и пятый биты, напишите BITFIELD$ = = MID$(BITSTRING,4,2). Затем определите десятичное значение (от 0 до 3), хранящееся в поле, используя процедуру обратного преобразования:

```
100 BITFIELD$ = MID$(BITPATTERN$,4,2): GOSUB 2000
                                       'вырезаем часть двоичной строки
110 PRINT DECVALUE                     'печатаем десятичное значение поля
.
.
2000 '''преобразуем строку 1 и 0 в десятичное число
2010 DECVALUE = 0                      'чистим переменную
2020 FOR N = 1 TO LEN(BITFIELD$)       'повторяем до конца поля
2030 DECVALUE = DECVALUE + VAL(MID$(BITFIELD$,N,1)*2^(N-1)
2040 NEXT
2050 RETURN
```

Приложение В. Основные сведения об языке ассемблера

Читатель, не знакомый с языком ассемблера, скоро поймет, что многие программистские трюки не могут быть достигнуты другими средствами. Хотя изучение языка ассемблера требует отдельной книги, в этом приложении приводятся основные понятия, которые помогут новичкам разобраться в примерах, написанных на этом языке. Внимательный просмотр разделов, посвященных среднему и низкому уровням, даст вам возможность получить представление о том, как работает ассемблер, после чего намного легче изучить разные частные вопросы. Здесь обсуждаются не все ассемблерные инструкции, встречающиеся в прог-

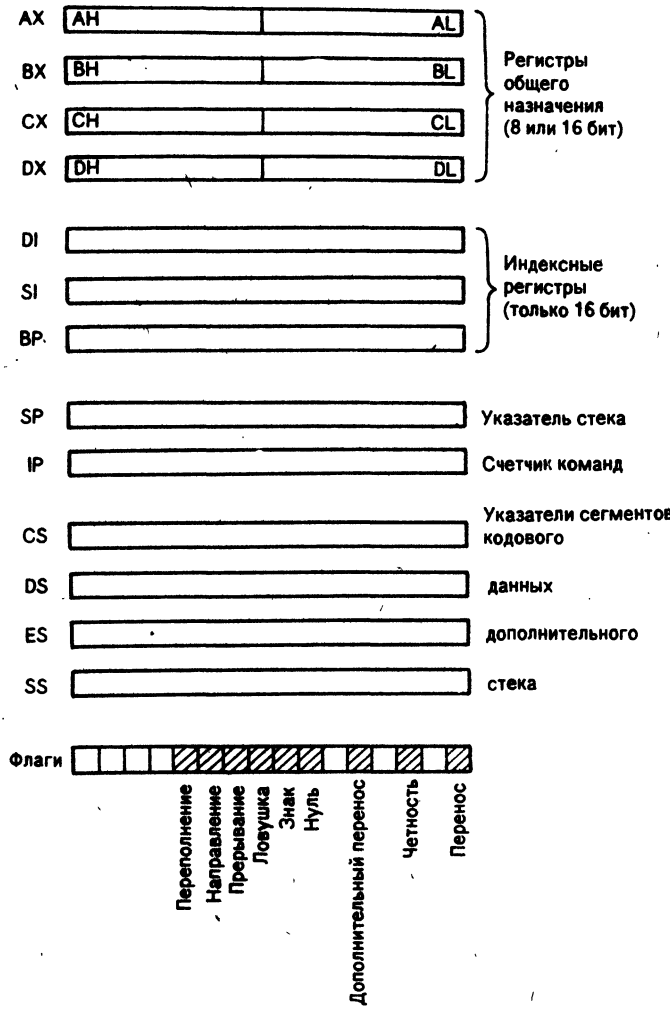


Рис. В.1

раммах, но вы обнаружите, что около 95 % инструкций, встреченных вами в программах, описаны здесь, а значение остальных может быть понято благодаря комментариям к программам.

Микропроцессор 8088 имеет 13 16-разрядных регистров, у каждого из которых есть свои функции. В то время как в языках высокого уровня вы можете поместить два числа в переменные, а затем сложить эти переменные, то в языке ассемблера эти числа помещаются в регистры микропроцессора, а затем складываются значения, содержащиеся в регистрах. Все операции в языке ассемблера состоят в обмене данных с регистрами, а затем выполнении операций на регистрах, таких, как изменение отдельных битов, выполнение арифметических операций и т.д. Одной из причин высокой эффективности языка ассемблера является хранение данных в регистрах микропроцессора; компиляторы имеют тенденцию возвращать все значения в память после выполнения операции, а доступ к памяти требует большого времени. На рис. В.1 показаны 13 регистров микропроцессоров 8088 и 80286 (последний имеет дополнительные средства для многозадачной работы, которые мы не будем здесь рассматривать).

Регистры AX, BX, CX и DX являются регистрами общего назначения. Их особенность состоит в том, что операции могут производиться не только над содержимым всего регистра, но также и над половиной. Каждый из четырех регистров делится на старшую и младшую части, например, AH обозначает старшую половину регистра AX, а AL - младшую. Точно так же ассемблерная программа может иметь доступ к BH, BL, CH, CL, DH и DL. Это свойство очень полезно, поскольку часто программе приходится работать с байтовыми величинами. Регистры BP, SI и DI также достаточно удобны, хотя они могут принимать только 16-битовые значения. Каждый бит регистра флагов сообщает о соответствующем статусе процессора, например, о том, что при выполнении арифметической операции был перенос за разрядную сетку.

В общем случае значения помещаются в регистры с помощью инструкции MOV. MOV AX,BX пересылает содержимое регистра BX в AX, затирая ранее содержащееся в AX значение. MOV AH,BL приводит к пересылке байта из регистра в регистр, но MOV AX,BL - недопустимая инструкция, так как значения должны иметь одинаковый размер. Инструкция MOV может также передавать значения из памяти, например, MOV AX,ACT_NUM. Здесь ACT_NUM - имя переменной, которую создал программист, совсем как в языке высокого уровня. Переменная создается оператором вида ACT_NUM DW 0. Этот оператор оставляет место для слова (двух байтов), присваивая им значение 0. Другие допустимые символы в этом операторе это DD (для двойного слова) и DB (для байта или строк). Ассемблер следит за адресами переменных, поэтому при ассемблировании оператора MOV AX,ACT_NUM имя переменной заменяется на ее адрес.

Работа с именами переменных - самый простой способ идентификации данных в программах на языке ассемблера. Но имеются различные способы хитрой адресации, которые позволяют программе хранить массивы или использовать указатели. Например, MOV AX,[BX][SI] посылает в AX значение, которое содержится по смещению, равному сумме значений регистров BX и SI. Но от чего отсчитывать смещение? Ответ заключается в том, что все данные собраны в одну часть программы, а весь исполняемый код - в другую. Часть, отведенная под данные, называется сегментом данных, а под программу - кодовым сегментом. Все переменные, отведенные для хранения данных, адресуются через смещение относительно начала сегмента данных.

Позиция в памяти, с которой начинается сегмент данных, хранится в регистре DS, одном из четырех сегментных регистров. Как и все остальные регистры микропроцессора, он 16-разрядный, поэтому не может содержать числа, большие чем 65535. Каким же образом сегмент даных может указывать на ячейки памяти, расположенные в верхней части мегабайтного адресного пространства? Ответ состоит в том, что сегментные регистры автоматически умножаются на 16, а результат указывает на место в памяти, с которого начинается сегмент. Таким образом, сегменты всегда выравнены на 16-байтовую границу. После того как сегмент установлен, все остальные регистры могут содержать смещения, указывающие на любой из следующих 65535 байт. Регистр дополнительного сегмента (ES) также используется для указания на данные, хранящиеся в памяти.

Среди ассемблерных инструкций, которые вы часто будете встречать в этой книге, есть инструкции загрузки сегментных и относительных адресов переменных. MOV AX,SEG ACT_NUM помещает значение сегментного регистра, в котором расположен ACT_NUM в AX, а впоследствии это значение будет переслано в DS. MOV BX,OFFSET ACT_NUM помещает в BX смещение переменной ACT_NUM в сегменте данных. После выполнения этих операций DS:BX будут указывать на ACT_NUM. Если ACT_NUM является одномерным массивом, то для указания на определенный элемент массива может использоваться добавочное смещение. Вы часто будете встречать также инструкцию LEA, предоставляющую другой способ загрузки смещения.

Кодовый сегмент содержит последовательность машинных инструкций, составляющих программу. Например, инструкция MOV существует в виде нескольких байтов машинного кода, значение байтов которого определяет, в какой регистр идет пересылка и откуда. Регистр IP (счетчик команд) содержит величину смещения, которая указывает на ту инструкцию в кодовом сегменте, которая сейчас должна выполняться. После выполнения инструк-

ции IP увеличивается таким образом, чтобы он указывал на следующую инструкцию. В простейшей программе счетчик команд будет передвигаться от первого байта кодового сегмента к последнему, где программа и завершится. Но, как и другие программы, программа на языке ассемблера может быть разбита на процедуры (подпрограммы), поэтому счетчик команд может прыгать из одного места кодового сегмента в другое.

Когда счетчик команд прыгает в другое место кодового сегмента, то его старое значение должно быть запомнено, с тем чтобы можно было вернуться в нужное место, как это делает оператор RETURN в Бейсике, возвращая управление в то место, откуда была вызвана процедура. В языке ассемблера процедуре присваивается имя, например COMB_DAT, и оператор CALL COMB_DAT передает управление в процедуру. Процедура завершается инструкцией RET (возврат). При вызове процедуры процессор запоминает текущее значение счетчика команд, складывая его в стек.

Стек - это область, используемая для временного хранения данных. После завершения процедуры старое значение счетчика команд берется из стека и выполнение программы продолжается. Стек также содержится в отдельном сегменте, который, совершенно естественно, называется сегментом стека. Ему соответствует сегментный регистр SS. В регистре SP хранится указатель стека, который всегда указывает на вершину стека и изменяется при засылке на стек и выборке из стека.

На первый взгляд стек кажется достаточно неуклюжим способом хранения информации, но у него есть два преимущества. Во-первых, доступ к его содержимому намного быстрее, чем к переменным, хранящимся в памяти, а во-вторых, стек может использоваться и для других целей. Он может хранить адреса возврата из процедуры, вложенной в другую процедуру. Впоследствии то же самое пространство может использоваться программистом для хранения данных, которые должны сейчас обрабатываться, но для которых не хватает места в регистрах микропроцессора. Программа складывает содержимое регистра в стек командой PUSH, а позднее забирает его оттуда командой POP. В ассемблерных программах, приведенных в этой книге, вы не раз встретитесь с инструкциями типа PUSH BX и POP DX. Неправильный порядок обмена данными со стеком - лучший способ привести ассемблерную программу к сбою.

После того как программист на языке ассемблера установил три сегментных регистра (CS, DS и SS) и загрузил данные в регистры микропроцессора, в его распоряжение предоставляется широкий набор встроенных средств, с помощью которых процессор может облегчить работу. Вот наиболее распространенные из них:

ADD	AX,BX	Прибавляет BX к AX. Существует еще инструкция вычитания (SUB), а также варианты обеих этих инструкций.
MUL	BL	Умножает BL на AX. Имеется еще инструкция деления (DIV), а также варианты обеих этих инструкций.
INC	BL	Увеличивает BL на 1. Имеется также инструкция уменьшения (DEC).
LOOP	XXX	Возвращает программу назад к строке, помеченной XXX, повторяя процесс столько раз, какое число содержится в CX (аналогично инструкции FOR .. TO .. NEXT в Бейсике).
OR	AL,BL	Выполняет операцию логического ИЛИ над содержимым регистров AL и BL, причем результат помещается в AL. Имеются также инструкции AND, XOR и NOT.
SHL	AX,1	Сдвигает все биты, содержащиеся в AX, на одну позицию влево. Это эквивалентно умножению содержимого AX на 2. Другие инструкции сдвигают биты вправо или осуществляют циклический сдвиг. Все они необходимы для битовых операций, таких, как установка точек экрана.
IN	AL,DX	Помещает в AX байт, обнаруженный в порте, адрес которого указан в DX. Имеется также инструкция OUT.
JMP		Передает управление в другое место программы, как инструкция GOTO в Бейсике. JMP YYY передает управление на строку программы, имеющую метку YYY.
CMP	AL,BL	Сравнивает содержимое AL и BL. За инструкцией CMP обычно следует инструкция условного перехода. Например, если за инструкцией CMP следует инструкция JGE, то переход произойдет, только если BL больше или равно AL. Инструкция CMP достигает того же результата, что и инструкция IF .. THEN в Бейсике (на самом деле инструкция IF .. THEN переводится интерпретатором Бейсика в инструкцию CMP).
TEST	AL,BL	Проверяет, есть ли среди битов, установленных в BL, такие, которые установлены также и в AL. За этой инструкцией обычно следует команда условного перехода, так же как за CMP. TEST очень полезен при проверке статусных битов (битовые операции очень просто реализуются в языке ассемблера).
MOVS		Пересылает строку, длина которой содержится в CX, с места, указываемого SI, на место, указываемое DI. Имеется еще несколько других инструкций, связанных с пересылкой и поиском строк.

Язык ассемблера обеспечивает несколько вариантов этих инструкций, а также ряд других специальных инструкций. К тому же имеется целый класс инструкций, называемых псевдооператорами, которые помещаются в текст программы с целью указания ассемблеру, как обрабатывать данную программу. Например, один из типов псевдооператоров автоматически вставляет часто использу-

емый кусок кода по всей программе. Такая порция кода называется макросом, и именно это свойство ассемблера дало ему название "макроассемблер".

И наконец, ассемблер предоставляет возможность, которой завидуют (или, по крайней мере, должны завидовать) все, кто программирует только на языках высокого уровня. Имеется в виду возможность оптимальным образом использовать прерывания операционной системы. Ведь это не что иное, как готовые процедуры. Однако вместо того, чтобы вызывать их по CALL, они вызываются инструкцией INT. INT21H вызывает прерывание с шестнадцатиричным номером 21. Имеется ряд таких прерываний, как в базовой системе ввода/вывода ПЗУ, так и в операционной системе, причем некоторые из этих процедур необычайно мощны. На самом деле некоторые из них настолько тесно связаны с системой, что вы практически не можете сами написать эквивалентную процедуру. Языки высокого уровня позволяют использовать многие из этих прерываний. Они применяют их для вывода на экран, приема ввода с клавиатуры и доступа к дискам. Но многие действительно полезные прерывания игнорируются языками высокого уровня, например, такие, которые позволяют запустить из одной программы другую. Некоторые трансляторы (такие, как Lattice C или Turbo Pascal) предоставляют доступ к этим прерываниям, если вы знаете, как их подготовить, и вы можете использовать для этой цели разделы среднего уровня данной книги.

Перед вызовом прерывания некоторая информация должна быть помещена в регистры процессора. Например, прерывание, вертикально сдвигающее экран, должно знать размеры сдвигаемого окна, число строк, на которое его надо сдвинуть, и т.д. Эти значения часто называют входными регистрами. Снова и снова вы будете встречать слова "при входе BX должен содержать ...", описывающие спецификацию входных регистров. Аналогично при возврате из прерывания некоторые регистры возвращают значения или статусную информацию. Они называются выходными регистрами, и мы описываем их словами "при выходе AX содержит ...". Зачастую одно прерывание содержит много функций. В частности, операционная система поместила практически все свои возможности в прерывание 21H. Поэтому при вызове прерывания необходимо указывать номер функции. Все прерывания (как BIOS, так и DOS) передают номер функции в AH (иногда в AL содержится номер подфункции).

Все ранее сказанное в основном служит только для того, чтобы дать вам первое представление о предмете. Но если вы будете внимательно просматривать простейшие примеры, содержащиеся в этой книге, то поймете стоящую за ними логику. Язык ассемблера имеет репутацию трудного языка. Но это утверждение не имеет за

собой веских оснований. Существует достаточно сложностей и в языках высокого уровня. И если ошибки в ассемблерной программе бывает очень сложно обнаружить, то в основном это связано с тем, что сам текст программы намного длиннее, чем эквивалентный текст на языке высокого уровня (однако ассемблерный код намного компактнее). В настоящее время многие профессионалы пишут программы на языке Си, затем анализируют эффективность и переписывают критические кусочки программы, которые расходуют много времени, на языке ассемблера. Невозможность написания таких ассемблерных процедур может иногда свести усилия программиста к нулю. Поэтому найдите хороший учебник для начинающих по ассемблеру и приступайте! Возможно, самой большой наградой для вас станет момент, когда вы, наконец, действительно станете понимать, как же работает компьютер.

Приложение Г. Включение ассемблерных процедур в программы на Бейсике

Процедуры на языке ассемблера состоят из строк байтов машинного кода. При выполнении процедуры Бейсик передает управление из последовательности инструкций, составляющих программу на Бейсике, в то место, где хранятся инструкции, которые могут быть декодированы в последовательность инструкций языка ассемблера. При завершении ассемблерной процедуры управление возвращается в то место бейсиковской программы, откуда была вызвана процедура.

В этой книге ассемблерные процедуры, используемые в программах на Бейсике, приведены в двух видах. В обоих видах они включены в программу, а не хранятся как отдельный дисковый файл. При первом способе требуется, чтобы коды процедуры находились в отдельном месте в памяти, а при втором, менее принятом, этого не требуется.

В первом способе процедура помещается в операторы DATA, и программа пересылается в неиспользуемую часть памяти, а затем вызывается оператором CALL. Надо позаботиться о том, чтобы код процедуры не накладывался на какие-либо данные, и наоборот. Обычное решение этой проблемы состоит в том, что процедура помещается в те адреса памяти, к которым Бейсик не может полу-

чить доступ. Поскольку интерпретатор Бейсика не может иметь доступ за пределы 64K, то для системы, скажем, с памятью 256K, нужно поместить процедуру в старшие 64K. Для систем с памятью 128K вы должны вычислить, сколько памяти требуется операционной системе, Бейсику и драйверам устройств. Допустимо, чтобы они занимали 25K плюс 64K, используемых Бейсиком. В системах с 64K применяйте при старте команду CLEAR, которая ограничивает объем памяти, доступный для Бейсика. CLEAR,n ограничивает Бейсик n байтами. Затем поместите процедуру в самые верхние адреса памяти.

Для указания начала области, куда будет помещена процедура, используйте оператор DEF SEG, а затем с помощью оператора READ считываются байты процедуры и помещаются в память до тех пор, пока вся процедура не будет помещена на место. Например:

```
100 DATA &Hxx, &Hxx, &Hxx, &Hxx, &Hxx   '10-байтовая процедура
110 DATA &Hxx, &Hxx, &Hxx, &Hxx, &Hxx
.
.
300 '''помещаем процедуру в память
310 DEF SEG = &H3000                    'указываем на область памяти
320 FOR N = 0 TO 9                      'для каждого из 10 байт
330 READ Q                              'читаем байт данных
340 POKE N,Q                            'помещаем его в память
350 NEXT
```

После того как процедура загружена в память и вы хотите ее использовать, необходимо, чтобы последний оператор DEF SEG указывал на начало процедуры. Затем присвойте целой переменной значение 0 и напишите оператор CALL с именем этой переменной. Если процедуре передаются параметры, то они должны быть указаны в скобках в конце оператора CALL. Например:

```
500 DEF SEG = &H3000                    'указываем на начало процедуры
510 DOGS = 12                           'у нее 3 параметра
520 CATS = 44
530 POSS = 1
540 CAS = 0                             'начинаем выполнение с 1-го байта
550 CALL CAS(DOGS,CATS,POSS)            'выполняем процедуру
```

Имеется намного более простой и экономичный способ создания ассемблерных процедур, который избегает проблемы распределения памяти. Надо просто создать процедуру в виде строковой переменной внутри программы. Каждый байт может быть закоди-

рован с помощью CHR$. Затем используйте функцию VARPTR для определения положения этой строки в памяти. Смещение, по которому находится эта переменная, хранится в двух байтах, следующих за тем, на который укажет VARPTR (в первом байте содержится длина строки). Затем этот адрес используется для вызова процедуры. Обратите внимание на способ применения оператора DEF SEG для указания на сегмент данных Бейсика, с тем чтобы полученное смещение указывало на адрес строки для оператора CALL. Например:

```
100 DEF SEG                          'устанавливаем сегмент на данные Бейсика
110 X$ = "CHR$(B4) + ..."            'код процедуры
120 Y = VARPTR(X$)                   'получаем дескриптор строки
130 Z = PEEK(Y + 1) + PEEK(Y + 2)*256        'вычисляем ее адрес
140 CALL Z
```

Многие значения, выражаемые через CHR$(), могут быть представлены и в виде символов ASCII. Вы можете писать ROUT = CHR$(12) + "AB" вместо ROUT = CHR$(12) + + CHR$(65) + CHR$(66). На самом деле большинство символов ASCII могут вводиться путем нажатия клавиши Alt, набора номера кода на дополнительной клавиатуре, а затем отпускания клавиши Alt. Однако для наших целей коды от 0 до 31 не могут быть введены таким образом.

Приложение Д. Использование драйвера устройства ANSI.SYS

ANSI.SYS - это небольшая программа, входящая в состав операционной системы, которая может быть загружена в память, с тем чтобы увеличить возможности MS-DOS. Она не сделана частью COMMAND.COM с целью экономии памяти, когда в ней нет необходимости. Средства, предоставляемые ANSI.SYS, могут быть использованы для удобства программирования, но они могут также служить средством достижения некоторой программной совместимости с машинами, несовместимыми с IBM PC, но работающими в MS-DOS. Этот драйвер не предоставляет никаких добавочных возможностей, которых нельзя было бы добиться другим образом, но он делает управление клавиатурой и терминалом более простым (и обычно более медленным). Все

свойства драйвера ANSI.SYS описаны в этой книге под соответствующим заголовком.

ANSI.SYS может быть загружен только во время загрузки операционной системы. Начиная с версии 2.0, система автоматически ищет файл CONFIG.SYS, так же как и файл AUTOEXEC.BAT. Файл CONFIG.SYS содержит различные параметры, такие, как число создаваемых буферов для файлов. Но он содержит также и имена тех драйверов устройств, которые должны быть загружены и включены в COMMAND.COM. ANSI.SYS как раз и является таким драйвером. Надо просто включить в этот файл строку DEVICE = ANSI.SYS. Она может быть единственной строкой в файле. Для его создания можно воспользоваться командой COPY. Надо просто ввести с терминала такие строки:

```
COPY CON: CONFIG.SYS «CR»
DEVICE = ANSI.SYS «CR»
«F6» «CR»
```

Нажатие клавиши F6 записывает символ Ctrl-Z (ASCII 26), отмечающий конец файла.

Приложение Е. Набор инструкций микропроцессора 8088

Число тактов, которое надо добавить для вычисления эффективного адреса, содержится в следующей таблице:

	Компоненты адреса	Операнды	Такты
(а)	база или индекс	[BX],[BP],[DI],[SI]	5
(б)	смещение	метка или смещение	6
(в)	база + индекс	[BX][SI], [BX][DI]	7
		[BP][SI], [BP][DI]	8
(г)	смещение + база или индекс	[BX],[BP],[DI],[SI] + смещ.	9
(д)	смещение + база + индекс	[BX][SI],[BX][DI] + смещ.	11
		[BP][SI],[BP][DI] + смещ.	12

Необходимо добавить также 2 такта при пересечении сегмента.
от времена инструкций:

Инструкция	Такты	Байты
AAA	4	1
AAD	60	2
AAM	83	1
AAS	4	1
ADC регистр, регистр	3	2
ADC регистр, память	9(13) + EA	2-4
ADC память, регистр	16(24) + EA	2-4
ADC регистр, значение	4	3-4
ADC память, значение	17(25) + EA	3-6
ADC аккумулятор, значение	4	2-3
ADD регистр, регистр	3	2
ADD регистр, память	9(13) + EA	2-4
ADD память, регистр	16(24) + EA	2-4
ADD регистр, значение	4	3-4
ADD память, значение	17(25) + EA	3-6
ADD аккумулятор, значение	4	2-3
AND регистр, регистр	3	2
AND регистр, память	9(13) + EA	2-4
AND память, регистр	16(24) + EA	2-4
AND регистр, значение	4	3-4
AND память, значение	17(25) + EA	3-6
AND аккумулятор, значение	4	2-3
CALL близкая процедура	23	3
CALL далекая процедура	36	5
CALL словный указатель в памяти	29 + EA	2-4
CALL словный регистр указатель	24	2
CALL двухсловный указатель в памяти	57 + EA	2-4
CBW	2	1
CLC	2	1
CLD	2	1
CLI	2	1
CMC	2	1
CMP регистр, регистр	3	2
CMP регистр, память	9(13) + EA	2-4
CMP память, регистр	9(13) + EA	2-4
CMP регистр, значение	4	3-4
CMP память, значение	10(14) + EA	3-6
CMP аккумулятор, значение	4	2-3
CMPS приемник, источник	22(30)	1
CMPS (REP) приемник, источник	9 + 22(30)/повтор	1
CWD	5	1
DAA	4	1
DAS	4	1
DEC словный регистр	2	1
DEC байтовый регистр	3	2
DEC память	15(23) + EA	2-4

Инструкция	Такты	Байты
DIV байтовый регистр	80-90	2
DIV словный регистр	144-162	2
DIV байт памяти	(86-96) + EA	2-4
DIV слово памяти	(154-172) + EA	2-4
ESC значение, память	8(12) + EA	2-4
ESC значение, регистр	2	2
HLT	2	1
IDIV байтовый регистр	101-112	2
IDIV словный регистр	165-185	2
IDIV байт памяти	(107-118) + EA	2-4
IDIV слово памяти	(175-194) + EA	2-4
IMUL байтовый регистр	80-98	2
IMUL словный регистр	128-154	2
IMUL байт памяти	(86-104) + EA	2-4
IMUL слово памяти	(138-164) + EA	2-4
IN аккумулятор, байт значения	10(14)	2
IN аккумулятор, DX	8(12)	1
INC словный регистр	2	1
INC байтовый регистр	3	2
INC память	15(23) + EA	2-4
INT 3	52	1
INT значение байта, отличное от 3	51	2
INTO	53 или 4	1
IRET	32	1
JCXZ короткая метка	18 или 6	2
JMP короткая метка	15	2
JMP близкая метка	15	3
JMP далекая метка	15	5
Jxxx короткая метка	16 или 4	2
LAHF	4	1
LDS словный регистр, двойное слово памяти	24 + EA	2-4
LEA словный регистр, слово памяти	2 + EA	2-4
LES словный регистр, двойное слово памяти	24 + EA	2-4
LOCK	2	1
LODS строка-источник	12(16)	1
LODS (REP) строка-источник	9 + 13(17)/повтор	1
LOOP короткая метка	17 или 5	2
LOOPE короткая метка	18 или 6	2
LOOPNE короткая метка	19 или 5	2
LOOPNZ короткая метка	19 или 5	2
LOOPZ короткая метка	18 или 6	2
MOV память, аккумулятор	10(14)	3
MOV аккумулятор, память	10(14)	3
MOV регистр, регистр	2	2
MOV регистр, память	8(12) + EA	2-4
MOV память, регистр	9(13) + EA	2-4
MOV регистр, значение	4	2-3
MOV значение, регистр	10(14) + EA	3
MOV сегментный регистр, словный регистр	2	2
MOV сегментный регистр, слово памяти	8(12) + EA	2-4

Инструкция	Такты	Байты
MOV словный регистр, сегментный регистр	2	2
MOV слово памяти, сегментный регистр	9(13) + EA	2-4
MOVS приемник, источник	18(26)	1
MOVS (REP) приемник, источник	9 + 17(25)/повтор	1
MUL байтовый регистр	70-77	2
MUL словный регистр	118-133	2
MUL байт памяти	(76-83) + EA	2-4
MUL слово памяти	(128-143) + EA	2-4
NEG регистр	3	2
NEG память	16(24) + EA	2-4
NOP	3	1
NOT регистр	3	2
NOT память	16(24) + EA	2-4
OR регистр, регистр	3	2
OR регистр, память	9(13) + EA	2-4
OR память, регистр	16(24) + EA	2-4
OR регистр, значение	4	3-4
OR память, значение	17(25) + EA	3-6
OR аккумулятор, значение	4	2-3
OUT байт значения, аккумулятор	10(14)	2
OUT DX, аккумулятор	8(12)	1
POP регистр	12	1
POP сегментный регистр	12	1
POP память	25 + EA	2-4
POPF	12	1
PUSH регистр	15	1
PUSH сегментный регистр	14	1
PUSH память	24 + EA	2-4
PUSHF	14	1
RCL регистр, 1	2	2
RCL регистр, CL	8 + 4/бит	2
RCL память, 1	15(23) + EA	2
RCL память, CL	20(28) + EA + 4/бит	2
RCR регистр, 1	2	2
RCR регистр, CL	8 + 4/бит	2
RCR память, 1	15(23) + EA	2
RCR память, CL	20(28) + EA + 4/бит	2
REP	2	1
REPE	2	1
REPNE	2	1
REPZ	2	1
REPNZ	2	1
RET (внутрисегментный, без POP)	20	1
RET (внутрисегментный, с POP)	24	3
RET (межсегментный, без POP)	32	1
RET (межсегментный, с POP)	31	3
ROL регистр, 1	2	2
ROL регистр, CL	8 + 4/бит	2
ROL память, 1	15(23) + EA	2
ROL память, CL	20(28) + EA + 4/бит	2

Инструкция	Такты	Байты
ROR регистр, 1	2	2
ROR регистр, CL	8 + 4/бит	2
ROR память, 1	15(23) + EA	2
ROR память, CL	20(28) + EA + 4/бит	2
SAHF	4	1
SAL регистр, 1	2	2
SAL регистр,CL	8 + 4/бит	2
SAL память, 1	15(23) + EA	2
SAL память, CL	20(28) + EA + 4/бит	2
SAR регистр, 1	2	2
SAR регистр, CL	8 + 4/бит	2
SAR память, 1	15(23) + EA	2
SAR память, CL	20(28) + EA + 4/бит	2
SBB регистр, регистр	3	2
SBB регистр, память	9(13) + EA	2-4
SBB память, регистр	16(24) + EA	2-4
SBB регистр, значение	4	3-4
SBB память, значение	17(25) + EA	3-6
SBB аккумулятор, значение	4	2-3
SCAS приемник	15(19)	1
SCAS (REP) приемник	9 + 15(19)/повтор	1
SHL регистр, 1	2	2
SHL регистр, CL	8 + 4/бит	2
SHL память, 1	15(23) + EA	2-4
SHL память, CL	20(28) + EA + 4/бит	2-4
SHR регистр, 1	2	2
SHR регистр, CL	8 + 4/бит	2
SHR память, 1	15(23) + EA	2-4
SHR память, CL	20(28) + EA + 4/бит	2-4
STC	2	1
STD	2	1
STI	2	1
STOS приемник	11(15)	1
STOS (REP) приемник	9 + 10(14)/повтор	1
SUB регистр, регистр	3	2
SUB регистр, память	9(13) + EA	2-4
SUB память, регистр	16(24) + EA	2-4
SUB регистр, значение	4	3-4
SUB память, значение	17(25) + EA	3-6
SUB AL, значение	4	2-3
TEST регистр, регистр	3	2
TEST регистр, память	9(13) + EA	2-4
TEST регистр, значение	5	3-4
TEST память, значение	11 + EA	3-6
TEST AL, значение	4	2-3
WAIT	3 + 5n	1
XCNG AL, словный регистр	3	1
XCNG память, регистр	17(25) + EA	2-4
XCNG регистр, регистр	4	2
XLAT таблица-источник	11	1

Инструкция	Такты	Байты
XOR регистр, регистр	3	2
XOR регистр, память	9(13) + EA	2-4
XOR память, регистр	16(24) + EA	2-4
XOR регистр, значение	4	3-4
XOR память, значение	17(25) + EA	3-6
XOR AL, значение	4	2

Приложение Ж. Набор инструкций микропроцессора 80286

Придерживаясь схемы, принятой в данной книге, здесь перечислены инструкции только для режимов реальной адресации. Более мощный микропроцессор 80286 не требует добавочного времени на вычисление эффективных адресов, нет также отличия в выполнении команд над байтовыми и словными переменными. Звездочка указывает, что вы должны добавить один такт, если при вычислении смещения суммируются три элемента. Буква m указывает число байтов следующей инструкции, а n - число повторений.

Инструкция	Такты	Байты
AAA	3	1
AAD	14	2
AAM	16	2
AAS	3	1
ADC регистр/память с регистром	2,7*	2
ADC значение с регистром/памятью	3,7*	3
ADC значение с аккумулятором	3	2
ADD регистр/память с регистром	2,7*	2
ADD значение с регистром/памятью	3,7*	3
ADD значение с аккумулятором	3	2
AND регистр/память с регистром	2,7*	2
AND значение с регистром/памятью	3,7*	3
AND значение с аккумулятором	3	2
CALL прямой внутри сегмента	7 + m	3
CALL косвенный через регистр/память внутри сегмента	7 + m,11 + m*	2
CALL прямой между сегментами	13 + m	5
CBW	2	1
CLC	2	1
CLD	2	1
CLI	3	1

Инструкция	Такты	Байты
CMC	2	1
CMP регистр/память с регистром	2,6*	2
CMP регистр с регистром/памятью	2,7*	2
CMP значение с регистром/памятью	3,6*	3
CMP значение с аккумулятором	3	2
CMPS повторенный CX раз	5 + 9n	2
CMPS байт или слово	8	1
CWD	2	1
DAA	3	1
DAS	3	1
DEC регистр/память	2,7*	2
DEC регистр	2	1
DIV байтовый регистр	14	2
DIV словный регистр	22	2
DIV байт памяти	17*	2
DIV слово памяти	25*	2
ESC	9-20*	2
HLT	2	1
IDIV байтовый регистр	17	2
IDIV словный регистр	25	2
IDIV байт памяти	20*	2
IDIV слово памяти	28*	2
IMUL байтовый регистр	13	2
IMUL словный регистр	21	2
IMUL байт памяти	16*	2
IMUL слово памяти	24*	2
IMUL умножение на целое значение	21,24*	3
IN фиксированный порт	5	2
IN переменный порт	5	1
INC регистр/память	2,7*	2
INC регистр	2	1
INS строка	5 + 4m	2
INS байт или слово	5	1
INT указанный тип	23 + m	2
INT тип 3	23 + m	1
INTO	24 + m или 3	1
IRET	17 + m	1
JCXZ	8 + m или 4	2
JMP короткий/длинный	7 + m	2
JMP прямой внутри сегмента	7 + m	2
JMP косвенный через регистр/память	7 + m,11 + m*	2
JMP прямой между сегментами	11 + m	2
Jxxx	7 + m или 3	2
LAHF	2	1
LDS	7*	2
LEA	3*	2
LES	7*	2
LOCK	0	1
LODS	5	1
LODS повторенный CX раз	5 + 4n	1

Инструкция	Такты	Байты
LOOP	8 + 4n или 4	2
LOOPZ/LOOPE	8 + 4n или 4	2
LOOPNZ/LOOPNE	8 + 4n или 4	2
MOV регистр в регистр/память	2,3*	2
MOV регистр/память в регистр	2,5*	2
MOV значение в регистр/память	2,3*	3
MOV значение в регистр	2	2
MOV память в аккумулятор	5	3
MOV аккумулятор в память	3	3
MOV регистр/память в сегментный регистр	2,5*	2
MOV сегментный регистр в регистр/память	2,3*	2
MOVS байт или слово	5	1
MOVS повторенное CX раз	5 + 4n	2
MUL байтовый регистр	13	2
MUL словный регистр	21	2
MUL байт памяти	16*	2
MUL слово памяти	24*	2
NEG	2	2
NOT регистр/память	2,7*	2
OR регистр/память с регистром	2,7*	2
OR значение с регистром/памятью	3,7*	3
OR значение с аккумулятором	3	2
OUT фиксированный порт	3	2
OUT переменный порт	3	1
OUTS строка	5 + 4m	2
OUTS байт или слово	5	1
POP память	5*	2
POP регистр	5	1
POP сегментный регистр	5	1
POPA	19	1
POPF	5	1
PUSH память	5*	2
PUSH регистр	3	1
PUSH сегментный регистр	3	1
PUSH значение	3	2
PUSHA	17	1
PUSHF	3	1
RCA регистр/память на 1	2,7*	2
RCA регистр/память на CX	5 + n, 8 + n*	2
RCA регистр/память на число	5 + n, 8 + n*	3
RCR регистр/память на 1	2,7*	2
RCR регистр/память на CX	5 + n, 8 + n*	2
RCR регистр/память на число	5 + n, 8 + n*	3
RET внутри сегмента	11 + m	1
RET внутри сегмента, добавляя значение к SP	11 + m	3
RET между сегментами	15 + m	1
RET между сегментами, добавляя значение к SP	15 + m	3
ROL регистр/память на 1	2,7*	2

Инструкция	Такты	Байты
ROL регистр/память на CX	5 + n, 8 + n*	2
ROL регистр/память на число	5 + n, 8 + n*	3
ROR регистр/память на 1	2,7*	2
ROR регистр/память на CX	5 + n, 8 + n*	2
ROR регистр/память на число	5 + n, 8 + n*	3
SAHF	2	1
SAL регистр/память на 1	2,7*	2
SAL регистр/память на CX	5 + n, 8 + n*	2
SAL регистр/память на число	5 + n, 8 + n*	3
SAR регистр/память на 1	2,7*	2
SAR регистр/память на CX	5 + n, 8 + n*	2
SAR регистр/память на число	5 + n, 8 + n*	3
SBB регистр/память с регистром	2,7*	2
SBB значение с регистром/памятью	3,7*	3
SBB значение с аккумулятором	3	2
SCAS повторенное CX раз	5 + 8n	2
SCAS байт или слово	7	1
SEG (переопределение сегмента)	0	1
SHL регистр/память на 1	2,7*	2
SHL регистр/память на CX	5 + n, 8 + n*	2
SHL регистр/память на число	5 + n, 8 + n*	3
STC	2	1
STD	2	1
STI	2	1
STOS повторенное CX раз	5 + 3n	2
STOS	3	1
SAL регистр/память на 1	2,7*	2
SAL регистр/память на CX	5 + n, 8 + n*	2
SAL регистр/память на число	5 + n, 8 + n*	3
SUB регистр/память с регистром	2,7*	2
SUB значение с регистром/памятью	3,7*	3
SUB значение с аккумулятором	3	2
TEST регистр/память с регистром	2,6*	2
TEST значение с регистром/памятью	3,6*	3
TEST значение с аккумулятором	3	2
WAIT	3	1
XCNG регистр/память с регистром	3,5*	2
XCNG регистр с аккумулятором	3	1
XLAT	5	1
XOR регистр/память с регистром	2,7*	2
XOR значение с регистром/памятью	3,7*	3
XOR значение с аккумулятором	3	2

Глоссарий

146818. Микросхема в AT, содержащая часы реального времени и информацию о конфигурации.

6845. Микросхема контроллера дисплея.

76496. Микросхема синтезатора звука PCjr.

765 (PD765). Микросхема контроллера НГМД.

8048. Микропроцессор клавиатуры.

8237. Микросхема прямого доступа к памяти (DMA).

8250. Микросхема коммуникационного адаптера.

8253. Микросхема программируемого таймера.

8255. Микросхема адаптера интерфейса с периферией.

8259. Микросхема контроллера прерываний.

8087. Микросхема математического сопроцессора на PC, XT и PCjr.

8088. Центральный процессор у PC, XT и PCjr.

80286. Центральный процессор у AT.

80287. Микросхема математического сопроцессора на AT.

AND(И). Логическая операция, сравнивающая значения двух цепочек битов и на этой основе создающая третье значение, где установлены только те биты, которые были установлены в обоих значениях компонентов.

ANSI.SYS. Драйвер устройства, поставляемый вместе с операционной системой, который способен выполнять многие функции BIOS. Он применяется для достижения программной совместимости с отличными от IBM PC машинами, использующими MS-DOS.

AUTOEXEC.BAT. Имя командного файла, который автоматически выполняется при загрузке системы.

B. Суффикс, обозначающий число, представленное в двоичном виде, например 10111011B. *См. приложение А.*

BIOS. Базовая система ввода/вывода, которая является частью операционной системы, постоянно хранящейся в ПЗУ машины.

CD. "Носитель обнаружен". *См. DCD.*

COM. Тип исполняемого файла, в котором привязка уже выполнена и поэтому все адреса правильно записаны в файле перед его загрузкой.

CONFIG.SYS. Имя специального файла, который система просматривает при загрузке. Этот файл содержит информацию о параметрах системы и драйверах устройств, которые должны быть установлены, что позволяет задать требуемую конфигурацию системы.

CPU. Центральный процессор, который выполняет инструкции, составляющие компьютерную программу. У всех IBM PC центральным процессором является микросхема 8088, за исключением PC AT, у которого процессором служит микросхема 80286.

CRC. *См. циклический контроль четности.*

CR/LF. Возврат каретки/перевод строки. Эта пара символов используется, чтобы вызвать перевод курсора или печатающей головки к началу следующей строки.

CRT. Электронно-лучевая трубка, т.е. видеодисплей.

CTS. Очистка посылки. Сигнал от модема порту коммуникации, индицирующий, что модем готов начать передачу данных. Он является частью процедуры установления связи.

DB. Термин языка ассемблера, указывающий, что объект данных имеет размер 1 байт, или что это строка, состоящая из однобайтовых кодов.

DCD. Обнаружен носитель данных. Сигнал от модема порту коммуникации, индицирующий, что установлена связь с другим модемом.

DD. Термин языка ассемблера, индицирующий, что объект данных имеет длину 4 байта.

DMA. *См. прямой доступ к памяти.*

DSR. Готовность набора данных. Сигнал коммуникационному порту от модема, индицирующий, что модем готов.

DTA. Область обмена с диском. Буфер, используемый для обмена с диском, при доступе методом управляющего блока файла.

DTA по умолчанию. Область переноса данных размером 128 байт, которая выделяется каждой программе и начинается со смещения 80H в префиксе программного сегмента.

DTR. Готовность приемника данных. Сигнал от коммуникационного порта к модему, индицирующий, что компьютер готов.

DW. Термин языка ассемблера, указывающий, что объект данных имеет длину 2 байта.

EOF. Сокращение для "конца файла".

Esc-последовательность. Управляющая строка, начинающаяся с символа Esc (ASCII 27). Например, большинство управляющих команд принтера выполняется с помощью Esc-последовательностей.

EXE. Исполняемый файл, который требует привязки при загрузке. Не все адреса программы могут быть установлены до тех пор, пока неизвестно ее положение в памяти. EXE-файлы имеют заголовок, который содержит информацию об этой привязке. Эти файлы загружаются немного дольше и требуют больше места на диске, чем файлы типа COM.

EXEC. Функция операционной системы, позволяющая программе запустить другую программу. Она может также загружать оверлеи.

FAT. *См. таблица размещения файлов.*

FCB. *См. управляющий блок файла.*

H. Суффикс, обозначающий число, представленное в шестнадцатиричном виде, например 0D3H. *См. приложение А.*

IOCTL. Управление вводом/выводом. Этот механизм, предоставляемый системой, позволяет программе взаимодействовать с драйвером устройства, прямо посылая и получая управляющие строки, а не включая их в поток данных, посылаемых драйверу устройства.

IRQ. Сокращение для "запроса на прерывание". Используется при ссылке на маскируемые аппаратные прерывания.

LSB. Младший бит или младший байт.

MSB. Старший бит или старший байт.

OR (ИЛИ). Логическая операция, сравнивающая значения двух цепочек битов и создающая третье значение, у которого

установлены все биты, установленные хотя бы у одного из компонентов.

PSP. *См. префикс программного сегмента.*

RI. Индикатор звонка. Сигнал от модема с автоответчиком порту коммуникации, сообщающий, что телефон, с которым связался модем, звонит.

ROM-BIOS. *См. BIOS.*

RTS. Запрос на посылку. Сигнал от коммуникационного порта к модему, указывающий, что компьютер хочет, чтобы были посланы данные.

SETBLOCK. Функция операционной системы, которая сокращает или увеличивает область памяти, отведенной данной программе.

XON/XOFF. Метод установления связи при последовательной связи, который использует коды ASCII 17 и 19 для сигнала передающей станции, что она должна соответственно возобновить или приостановить передачу. Они используются, когда данные поступают слишком быстро и программа не успевает их обработать.

XOR (исключающее ИЛИ). Логическая операция, сравнивающая битовые цепочки двух значений и создающая третье значение, где установлены только те биты, для которых только один из сравниваемых битов был установлен.

Абсолютные координаты. Координаты, указанные относительно центральной оси, а не относительно предыдущих используемых координат (относительные координаты).

Абсолютные сектора диска. Под "доступом к абсолютному сектору диска" понимается чтение сектора, занимающего определенное положение на диске.

Абсолютный адрес. Адрес памяти, выраженный в виде смещения относительно младшего адреса (0000:0000), а не относительно какого-либо определенного смещения в памяти (относительный адрес).

Адрес порта. Число в диапазоне от 0 до 65535, которое адресует порт. Адреса портов отделены от адресов памяти. Доступ к портам осуществляется с помощью инструкций IN и OUT в языке ассемблера и INP и OUT в Бейсике.

Адресация. Средство доступа к определенным ячейкам памяти за счет указания либо их абсолютного положения, либо относительного смещения.

Адресный регистр. Регистр одной из вспомогательных микросхем, который служит в качестве указателя на один из нескольких регистров данных микросхемы, доступ к которым осуществляется через один порт. Программа должна сначала индексировать регистр, посылая номер интересующего регистра в адресный регистр.

Аппаратное прерывание. Прерывание, вызываемое оборудованием, т.е. одним из периферийных устройств, микросхемой поддержки или самим процессором.

Аппаратный сдвиг экрана. Метод вертикального сдвига изображения на дисплее, основанный на изменении стартовой точки видеобуфера, а не на сдвиге содержимого буфера.

Асинхронная связь. Последовательный канал связи, в котором время между посылкой символов может быть переменным.

Ассемблер. Программа, преобразующая текст программы на языке ассемблера в машинный код.

Атрибут. Характеристика, приписываемая устройству или данным. Каждый символ текстового экрана имеет атрибуты, определяющие его цвет, интенсивность и т.д. Драйверы устройств имеют атрибуты, определяющие способ обработки данных, управляющие строки и т.д. Файлы могут иметь атрибуты, указывающие, что они являются скрытыми, только для чтения и т.д.

Атрибуты файла. Поле элемента каталога файлов, определяющее статус файла и делающее его обычным, скрытым, только для чтения и т.п.

Атрибуты цвета. Цепочки битов, хранимые в видеобуфере, которые определяют цвет определенной точки или символа на экране. Для монохромного и цветного адаптера эти атрибуты совпадают с системой кодовых номеров цвета. Однако для PCjr и EGA они относятся к номеру регистра палитры, а уже этот регистр содержит код цвета, с которым связан этот атрибут.

Базовый адрес. Младший из группы смежных адресов портов, через которые осуществляется доступ к периферийному устройству.

Байт атрибутов. Вообще говоря, байт, содержащий код, устанавливающий специальные характеристики среды, к которой он относится. Байт атрибутов файла (в дисковом каталоге) определяет статус скрытого файла, статус только для чтения и т.п. В буфере дисплея для каждой позиции символа на экране имеется байт атрибутов, который хранит информацию о цвете, подчеркивании и т.д.

Байт статуса. Ячейка памяти, содержащая цепочку битов, описывающую текущий статус устройства.

Бит четности. Дополнительный (9-й) бит, добавляемый к каждому байту памяти, чтобы проверять возможные ошибки при передаче. Биты четности присоединяются также к данным при последовательной коммуникации.

Битовая плоскость. В EGA видеобуфер разделен на четыре области, которые называются битовыми плоскостями 0-3. В режиме 16 цветов четыре плоскости параллельны, при этом 4 байта относятся к определенному адресу памяти (регистры задвижки определяют обмен данными между процессором и памятью дисплея). В некоторых случаях плоскости могут быть связаны в цепь, образуя одну или две большие плоскости.

Битовое поле. Когда байт или слово рассматриваются как цепочка битов, то некоторые биты, взятые вместе, могут хранить определенный элемент информации. Например, биты 0-3 байта атрибутов символа на дисплее образуют битовое поле, которое определяет основной цвет символа.

Битовые операции. Программные операции, читающие или изменяющие определенные биты данных.

Блок параметров. Группа переменных, создаваемая в памяти для хранения информации, используемой устройством или функцией операционной системы.

Блок прямого доступа. Блок записей, которые считываются или записываются за одну операцию с файлом прямого доступа при доступе к файлу методом управляющего блока файла.

Блочные устройства. Устройства, которые посылают и принимают данные порциями в блок. Дисковые накопители являются наиболее распространенными блочными устройствами.

Буфер. Область памяти, отводимая для хранения данных, которые будут передаваться от одной части компьютера к другой.

Буфер используется клавиатурой, то же самое относится и к дисковым накопителям и дисплею.

Буфер клавиатуры. 15-символьная циклическая очередь, в которую прерывание клавиатуры помещает вводимые символы.

Вектор. Четырехбайтовый адрес процедуры прерывания в памяти. Старшие два байта дают сегмент, а младшие два байта - смещение.

Видеобуфер. Область памяти, отведенная для хранения информации, выводимой на экран. Например, цветной графический адаптер использует буфер размером 16K. Микросхема управления дисплеем постоянно сканирует буфер, декодируя его содержимое и проектируя его на экран.

Возврат. Выражение "при возврате ..." относится к информации, которая будет содержаться в регистрах процессора после выполнения функции операционной системы.

Вход. Слова "при входе" обычно относятся к установке регистров процессора, которая должна быть сделана при выполнении функции операционной системы.

Главная запись загрузки. Запись начальной загрузки на жестком диске. Она содержит таблицу разделов, указывающую на различные разделы диска. Каждый из разделов содержит обычную запись начальной загрузки, которая инициирует загрузку соответствующей операционной системы.

Глобальный символ. Один из символов ? или *, когда они используются в системе для указания неопределенных символов в именах файлов.

Граница. Определенный интервал в памяти, в файле и т.д. Например, программы размещаются в памяти, выравненными на 16-байтовую границу. Это означает, что абсолютные адреса этих ячеек должны точно делиться на 16.

Дерево каталогов. Система подкаталогов, организованная как ветви дерева, в которой на каталоги первого уровня имеются ссылки в корневом каталоге, а те в свою очередь содержат ссылки на каталоги более низкого уровня.

Дескриптор файла. В Бейсике или другом языке высокого уровня под дескриптором файла понимается номер буфера, с которым данный файл открывается, т.е. как #1 или #3.

Добавочный сегмент. Область памяти, на которую указывает регистр процессора ES. Установка ES и DS (регистр сегмента

данных) часто используется совместно для переноса данных из одной части памяти в другую.

Дорожка. Кольцо на диске. Дискета емкостью 360K разделена на 40 дорожек, каждая из которых делится радиально на 9 секторов.

Драйвер устройства. Программная процедура, управляющая устройством, таким, как дисковый накопитель или принтер.

Заголовок. Блок параметров, помещаемый в начало программы, драйвера устройства или другого массива кода или данных. Заголовок содержит информацию о коде или данных, которая важна для их использования. Например, операционная система помещает 256-байтовый заголовок перед началом каждой загружаемой программы - префикс программного сегмента - и использует содержащуюся в нем информацию для работы с этой программой.

Заголовок запроса. Блок параметров, создаваемый системой для управления драйвером устройства.

Заголовок устройства. Начальная часть процедуры драйвера устройства, которая идентифицирует устройство.

Запись. Блок данных, указанного размера, являющийся единицей обмена данными с файлами.

Запись начальной загрузки. Короткая программа, помещаемая на диск в такой позиции, которая считывается с диска в первую очередь при загрузке системы. Эта программа дает компьютеру возможность загрузить части операционной системы.

Запрос системы. Символы, появляющиеся в начале командной строки, например, A> или B>.

Интерпретатор. Программа, которая переводит текст программы по одной инструкции, немедленно исполняя ее. Интерпретаторами являются программы BASIC.COM и BASICA.COM.

Кластер. Группа дисковых секторов, образующая основную единицу, которая используется при распределении дискового пространства.

Код. Набор выполняемых инструкций, составляющих программу, в отличие от данных, над которыми выполняются операции. Вообще говоря, кодом называется последовательность машинных инструкций, которые производит транслятор или ассемблер из текста программы.

Код доступа. Этот термин используется в Техническом руководстве по MS-DOS для номера подфункции, т.е. для кода одной из нескольких функций, которые могут выполняться данным прерыванием.

Код завершения. Код, передаваемый процессом-потомком процессу-родителю. Например, когда одна программа запускает другую, то код завершения может быть передан от потомка родителю при завершении задачи потомка. Эти коды могут определяться программистом.

Код нажатия. Тип скан-кода, который генерируется при нажатии клавиши (код освобождения выдается, когда соответствующая клавиша отпускается).

Код отпускания. Тип скан-кода, который генерируется при отпускании клавиши (код нажатия генерируется при нажатии клавиши).

Код ошибки. Кодовый номер, выдаваемый операционной системой для индикации определенного ошибочного условия.

Код палитры. Номер, соответствующий определенному цвету из доступного набора.

Код цвета. Число от 0 до 15, которое относится к одному из шестнадцати цветов дисплея. Дисплей EGA, присоединенный к улучшенному графическому адаптеру, может иметь 64 кода цвета (0-63).

Кодовый сегмент. Область памяти, хранящая программный код (другие сегменты хранят данные и стек).

Коды ASCII. Набор кодов от 0 до 127, соответствующих одному из 128 символов ASCII. IBM PC использует расширенный набор кодов ASCII, состоящий из 256 символов.

Командная строка. Строка на экране дисплея, принимающая управляющую информацию, такая, как строка, начинающаяся с запроса операционной системы.

Командный файл. Файл, содержащий список команд и программ DOS, которые будут автоматически вызываться в том порядке, в котором они записаны, либо порядок их выполнения может определяться условными операторами.

Коммуникационное прерывание. Аппаратное прерывание, вызываемое адаптером асинхронной связи. Оно может происходить при получении очередного символа по линии

связи, когда наступило время передавать следующий символ и т.п.

Компилятор. Программа, преобразующая текст программы на языке высокого уровня в файл, содержащий исполняемый машинный код (или иногда в промежуточный код, который затем исполняется интерпретатором).

Компоновщик. Программа, которая компонует вместе объектные модули программы, организуя их адреса таким образом, чтобы модули могли взаимодействовать. Даже программы, состоящие из одного модуля, должны быть скомпонованы, поскольку компоновщик создает также код привязки.

Конечная строка. Строка матрицы символов, на которой кончается изображение курсора. *См. начальная строка.*

Корневой каталог. Центральный каталог диска. Он расположен в фиксированном месте на диске. Он может содержать список файлов, метку тома и указатели на подкаталоги.

Критическая ошибка. Ошибка устройства, которая делает дальнейшее выполнение программы невозможным. При этом вызывается обработчик критических ошибок операционной системы.

Логический номер сектора. Вместо того чтобы указывать сектора диска как "сторона x, дорожка x, сектор x", используются логические номера секторов, которые определяют позицию сектора за счет последовательной нумерации секторов, начиная с внешней границы диска.

Маркировка. Термин, используемый для последовательного сигнала, когда он имеет высокий уровень, т.е. равен логической 1. В частности, говорят, что сигнал асинхронной связи маркирован в промежутки времени между передачей элементов данных.

Маска. Цепочка битов, определяющая, какие из битов второй цепочки являются активными. Например, определенные аппаратные прерывания запрещаются за счет установки битов в регистре маски микросхемы контроллера прерываний. При этом прерывание 4 маскируется цепочкой битов 00001000B.

Массив ворот дисплея. Микросхема видеосистемы PCjr, содержащая ряд управляющих и статусных регистров, включая регистры палитры.

Масштабный коэффициент. Отношение числа точек, занимающих одно и то же расстояние по вертикали и горизонтали на экране терминала или печатающем устройстве.

Машинная инструкция. Числовые коды, используемые процессором. Например, инструкция INT кодируется как CD, а последовательность CD 21 приводит к тому, что процессор выполняет прерывание 21H.

Машинный язык. Самый низкий уровень программирования, когда программист пишет инструкции непосредственно в двоичных кодах, используемых процессором. Программирование на языке ассемблера приводит к тем же результатам, с бо́льшими удобствами за счет создания двоичных кодов из мнемоники типа MOV или TEST.

Метка тома. Специальный элемент корневого каталога, обеспечивающий возможность идентифицировать диск 11-байтовым именем.

Метод дескриптора файлов. Метод доступа к файлам с использованием номера файла. Этот метод практически вытеснил ранее используемый метод доступа с помощью управляющего блока файла.

Метод управляющего блока файла. Набор функций операционной системы, позволяющий доступ к файлам посредством управляющего блока файла. Этот метод стал устаревшим после введения метода доступа с использованием дескриптора файлов.

Микросхема поддержки. Одна из многих микросхем, которая связывает процессор с другими частями компьютера или внешними устройствами. Наш словарь начинается со списка микросхем поддержки, обсуждаемых в данной книге.

Мировые координаты. Система координат экрана, определяемая программой, устанавливающей диапазон значений координат x и y, которые могут включать или не включать отрицательные значения. Например, левому и правому краю экрана могут быть присвоены координаты -100 и 100. Эти координаты накладываются на систему физических координат экрана, в которой левый верхний угол всегда определяется значениями x = 0 и y = 0 и в которой используются только положительные значения.

Начальная строка. Строка матрицы символов, на которой начинается изображение курсора. Например, для монохромного дисплея строка текста состоит из матрицы высотой в 14 строк, которые пронумерованы от 0 до 13. Для обычного курсора номер начальной строки 12, а конечной - 13.

Начальный кластер. Первый кластер, с которого файл записывается на диск. Элемент каталога файлов указывает на начальный кластер, а таблица размещения файлов хранит информацию о последующих кластерах, используемых файлом.

Номер записи. Число, определяющее позицию записи в файле, отсчитываемое от 0. В файле, содержащем записи длиной 10 байт, запись номер 5 относится к 50-59 байтам файла, даже если записи с меньшими номерами не вводились.

Номер записи прямого доступа. Номер, вводимый в поле записи прямого доступа управляющего блока файла. Последующие файловые операции преобразуют этот номер в номер текущего блока и текущей записи.

Номер параграфа. Номер, определяющий положение в памяти, основываясь на 16-байтовых единицах. Например, параграф номер 2 относится ко вторым 16 байтам памяти и когда указатель указывает на этот параграф, то он указывает на 17-й байт памяти.

Номер файла. Кодовый номер, возвращаемый системой, когда файл открывается с использованием метода дескриптора файлов. Этот номер впоследствии необходим для указания файла при дисковых операциях. Некоторые предопределенные номера идентифицируют дисплей, принтер и т.д.

Область данных BIOS. Область данных, начинающаяся с адреса 0040:0000, в которой BIOS хранит статусную информацию и буфер клавиатуры.

Область переноса данных. Буфер, используемый при доступе к файлам с помощью метода управляющего блока файла, который содержит данные, передаваемые на диск или с диска.

Обработка ошибок. Код, позволяющий программе передать управление специальной процедуре восстановления в случае возникновения критической ошибки при сбоях.

Обработчик критических ошибок. Прерывание системы, которое вызывается при возникновении критической ошибки. Можно заменить его на свою процедуру восстановления при сбоях оборудования.

Обработчик прерывания. Процедура прерывания. Этот термин более часто используется для аппаратных прерываний.

Обработчик прерывания устройства. Основная часть процедуры драйвера устройства; она содержит код, выполняющий основные функции драйвера.

Объектный модуль. Файл, содержащий машинный код, в котором еще не установлены относительные адреса. Компоновщик обрабатывает и объединяет объектные модули, создавая исполняемые файлы типа EXE или COM.

Оверлей. Подпрограмма, хранящаяся на диске до тех пор, пока она не потребуется головной программе. Она загружается в память поверх одной из частей вызывающей программы.

Ограничитель. Специальный символ, разделяющий элементы данных.

Операции в реальном времени. Программные операции, которые должны выполняться в определенный момент, а не тогда, когда компьютер окажется способным их выполнить. Мультипликация, сигналы тревоги и роботы используют работу в реальном времени.

Определение перерыва. Способность адаптера коммуникации распознавать длинную последовательность логических нулей. Это сигнализирует о том, что отдаленная станция хочет перерыва в связи.

Опрос. Управление периферийным устройством за счет постоянной проверки его статуса до тех пор, пока не произойдут желаемые изменения.

Орнамент. Заполнение области графического дисплея определенной картинкой, а не одним цветом.

Основной цвет. Цвет, которым символы или графические объекты выводятся на экран.

Отведение памяти. Отведение системой блока памяти для использования программой.

Относительный адрес. Адрес памяти, который указан в виде смещения относительно некоторой определенной точки памяти. Например, в COM-файлах переменные указываются адресами относительно начала программы.

Относительные координаты. Координаты, определяемые относительно последних используемых координат. В этом случае 3,5 указывает "3 вправо и 5 вверх", а -3,-5 - "3 влево и 5 вниз".

Отображение в память. Помещение данных, выводимых на дисплей, непосредственно в видеобуфер (откуда они проектируются на экран), вместо того чтобы использовать функции, предоставляемые операционной системой или языком высокого уровня.

Ошибка обрамления. Ошибка при последовательной связи, когда поток данных несинхронизован, т.е. биты данных, биты четности, стартовые и стоповые биты не идут в правильной последовательности.

Палитра. Набор цветов, доступных в определенном режиме дисплея.

Параграф. 16-байтовая единица памяти, которая начинается на границе, делящейся на 16 без остатка.

Параметр. Число, используемое для спецификации работы устройства, функции операционной системы или оператора языка программирования.

Переполнение. Переполнение происходит, когда данные в буфере или регистре стираются из-за поступления новых данных прежде, чем они были обработаны.

Подкаталог. Каталог, который ничем не отличается от корневого каталога, за исключением того, что он хранится на диске как файл, а не в абсолютных секторах диска. Корневой каталог может содержать элементы, указывающие на подкаталоги, а они в свою очередь могут содержать элементы, описывающие другие подкаталоги.

Подпрограмма на машинном языке. Подпрограмма, написанная на языке ассемблера, которая затем ассемблирована и включена в программу, написанную на языке высокого уровня. Такие подпрограммы обычно создаются для операций, которые часто повторяются и должны выполняться очень быстро. В зависимости от того,

используется ли транслятор или интерпретатор, машинные коды могут быть скомпонованы с программой, включены в программу отдельными строками или отдельно загружаться в память с диска.

Подтверждение. Сигнал ввода/вывода, индицирующий, что задача выполнена и оборудование готово начать выполнение новой задачи.

Подфункция. Одна из нескольких процедур, которые могут выполняться данной функцией операционной системы. В то время как номер функции всегда помещается в AH, номер подфункции надо поместить в AL перед выполнением прерывания.

Поле. Группа битов или байтов, отведенная для хранения определенного элемента данных.

Порт. Путь, по которому происходит обмен данными между процессором и микросхемами поддержки.

Порт A (порт B, порт C). Один из трех регистров, через которые программа получает доступ к микросхеме интерфейса с периферией 8255.

Прерывание. Прерывания - это программные процедуры, которые могут вызываться двумя способами. Аппаратные прерывания инициируются оборудованием, например, когда нажимается клавиша на клавиатуре, то это событие мгновенно обрабатывается процессором, который выполняет требуемые действия и возвращается к прерванной работе. Программные прерывания служат для выполнения стандартных потребностей программиста, таких, как посылка символа на экран или принтер. Они предоставляются операционной системой и начинают работать, когда программа явно обратится к ним.

Прерывание клавиатуры. Аппаратное прерывание, вызываемое, когда клавиша на клавиатуре нажимается или отпускается. Оно преобразует скан-коды, выдаваемые микропроцессором клавиатуры, в коды, используемые программами, и вставляет эти коды в буфер клавиатуры.

Прерывание принтера. Аппаратное прерывание, которое происходит, когда адаптер принтера посылает сигнал "не занят". Процедура прерывания обычно посылает на принтер следующий байт выводимых данных и возвращает

управление. Таким образом можно выводить файлы на печать в то время, когда компьютер занят другой задачей.

Прерывание таймера. Прерывание, инициируемое микросхемой таймера 8253 18,2 раза в секунду. Каждый раз это прерывание увеличивает счетчик времени суток BIOS.

Префикс программного сегмента. 256-байтовый заголовок, который система помещает перед исполняемыми файлами при их загрузке в память. Он содержит переменные, используемые MS-DOS для управления программой, а также место для управляющего блока файла и область переноса данных.

Привязка. Процесс, выполняемый системой при загрузке программ типа EXE. Система вычисляет базовые адреса (адреса сегментов), от которых будут отсчитываться все остальные адреса. Эти базовые адреса не могут быть установлены заранее, до загрузки программы, поскольку позиция программы в памяти до этого времени неизвестна. Программы типа COM не требуют привязки.

Программное прерывание. Прерывание, вызываемое инструкцией INT.

Пространство памяти. Область адресуемой памяти, к которой процессор может иметь доступ. Для микропроцессора 8088 адресуемое пространство равно приближенно одному миллиону байт.

Протокол. Система параметров и форматов данных, используемых устройством.

Процесс-потомок. Программа, запускаемая, когда другая программа (родитель) имеет управление.

Процесс-родитель. Программа, использующая другую программу (процесс-потомок).

Прямое отображение в память. *См. отображение в память.*

Прямой доступ к памяти. Способ осуществления очень быстрого обмена между периферийным устройством и памятью. Он особенно полезен при дисковых операциях. Этот метод использует специальную микросхему (которая отсутствует у PCjr).

Раздел. Область жесткого диска. Жесткий диск может быть разбит на разделы, с тем чтобы он использовался несколькими операционными системами.

Расширенный код. Код клавиши, используемый для идентификации нажатия этой клавиши (или комбинации клавиш), для которой нет соответствующего символа в наборе ASCII, такой, как функциональные клавиши или комбинации с клавишами «Ctrl» или «Alt». Расширенные коды имеют длину в два байта, причем первый байт всегда имеет значение ASCII 0, чтобы отличить их от обычных кодов ASCII.

Расширенный код ошибки. Начиная с версии 3.0 MS-DOS более подробные расширенные коды ошибки возвращаются при возникновении ошибки. Эти коды сообщают не только об ошибке, но и об ее типе, ее месте в оборудовании и возможных способах восстановления.

Расширенный управляющий блок файла. Управляющий блок файла, имеющий добавочное 7-байтовое поле заголовка, устанавливающее атрибуты файла.

Регистр. Часть микросхемы, в которой данные хранятся и над ними производятся операции. В IBM PC большинство регистров имеет размер 8 или 16 бит. Регистры процессора получают значения из памяти и хранят их, пока они складываются, умножаются и т.д. Регистры микросхемы управления дисплеем инициализируются данными, определяющими характеристики дисплея.

Регистр палитры. Один из 16 регистров EGA и PCjr, указывающий цвет, который будет выводиться на экран, когда соответствующий код цвета указан в видеобуфере.

Регистр статуса. Регистр ввода/вывода, содержащий цепочку битов, описывающую текущий статус устройства.

Регистры задвижки. У EGA имеется 4 однобайтовых регистра задвижки, которые хранят 4 байта данных, относящихся к определенному адресу видеобуфера. Когда процессор читает из буфера, то регистры задвижки заполняются, а когда процессор пишет в видеобуфер, то содержимое регистров задвижки пересылается в соответствующие ячейки памяти.

Резидентная программа. Программа, остающаяся в памяти после завершения. Система предохраняет ее от порчи другими загружаемыми программами, которые могут иметь доступ к содержащимся в данной программе процедурам через вектора прерывания.

Рукопожатие. Обмен предопределенными сигналами между двумя устройствами для установления связи между ними.

Связь в цепочку. У EGA видеопамять разделена на 4 битовые плоскости. Когда они объединяются в одну или две большие плоскости, то это называется связью в цепочку.

Сегмент. Область памяти размером 64K, созданная для хранения кода, данных или стека. Сегменты всегда выравнены на границу 16 байт, поскольку их адрес получается умножением содержимого сегментного регистра на 16.

Сегмент данных. Область памяти, содержащая данные программы. В языке ассемблера на эту область указывает регистр DS.

Сегмент стека. Область памяти, отводимая для хранения стека.

Сегментное значение. Число, определяющее положение в памяти в 16-байтовых единицах. То же, что и номер параграфа.

Сегментный адрес. То же, что и сегментное значение или номер параграфа.

Сегментный регистр. Один из четырех регистров процессора, указывающий на начальную позицию сегмента памяти. Значение этого регистра автоматически умножается на 16, с тем чтобы он указывал на одну из 16-байтовых границ мегабайтового адресного пространства процессора. Имена сегментных регистров CS (кодовый сегмент), DS (сегмент данных), SS (сегмент стека) и ES (добавочный сегмент).

Символьное устройство. Устройство, которое посылает или принимает данные по одному символу, такие, как принтер. Сравните с блочными устройствами, которые обмениваются данными блоками.

Синтаксический анализ. Разбиение текстовой строки на составляющие части. MS-DOS может, проанализировав информацию в командной строке, переформатировать ее для использования функциями доступа к файлу.

Синхронная связь. Последовательная связь, при которой приемная и передающая станции посылают и принимают сигналы со строго синхронизованной скоростью.

Системные часы. Кристалл, генерирующий импульсы заданной частоты, которая определяет работу всех устройств, в том числе и микросхемы таймера 8253.

Системный файл. Специальный статус, присваиваемый файлу посредством байта атрибутов. Он отмечает файлы, являющиеся частью операционной системы.

Скан-код. Кодовое число, посылаемое микропроцессором клавиатуры 8048 микросхеме интерфейса с периферией 8255 (или эквивалентной), которое сообщает, какая клавиша клавиатуры была нажата или отпущена. Прерывание клавиатуры преобразует скан-коды в коды ASCII или расширенные коды и устанавливает статус клавиш-переключателей.

Скорость обмена. Число битов в секунду, которое передается при обмене.

Скрытый файл. Статус, который может быть присвоен файлу установкой его байта атрибутов. Скрытые файлы не выводятся при выводе каталога файлов.

Слово. Вообще говоря, словом называется основная единица данных, используемая микропроцессором. В данной книге этот термин всегда относится к двухбайтовой величине.

Стартовый бит. При последовательной связи стартовый бит предшествует каждому слову данных. Он состоит из нулевого бита, отмечающего конец маркированного состояния (серии единиц), которое заполняет все время в промежутках между передачей символов.

Стек. Область памяти, используемая программой для временного хранения данных. Последний элемент, помещаемый в стек, забирается оттуда первым. Доступ к стеку более быстрый, чем к переменным.

Стоповый бит. При последовательной связи стоповые биты следуют за каждым словом данных. Они переводят коммуникационную линию в маркированное состояние и оставляют ее в этом состоянии на минимальное время, которое должно пройти, прежде чем можно послать следующее слово.

Страница. При работе с дисплеем страницей называется часть видеобуфера, хранящая данные для одного экрана. Можно переключать дисплей между страницами, с тем чтобы он выводил сначала содержимое одной страницы, а затем другой. Термин "страница" часто используют и для обозначения 256-байтового раздела памяти.

Стратегия устройства. Часть процедуры драйвера устройства, связывающая драйвер с заголовком запроса, который является блоком параметров, создаваемым системой для управления драйвером.

Строка ASCIIZ. То же, что и строка пути.

Строка окружения. Строка, состоящая из одной или более спецификаций, которым система следует при выполнении программы. Она может содержать конфигурационные команды, вводимые пользователем, такие, как BUFFERS или BREAK.

Строка пути. Строка, используемая для указания файла при доступе методом дескриптора файлов. Строка имеет тот же вид, что и при доступе на командном уровне системы. Она может начинаться с имени накопителя, может содержать имена подкаталогов, разделяемые обратной косой чертой, и должна завершаться байтом ASCII 0, отмечающим ее конец. Максимально допустимая длина строки 63 байта.

Счетчик времени суток. Переменная в области данных BIOS, которая постоянно увеличивается прерыванием таймера. Ее значение используется операционной системой для вычисления времени суток.

Счетчик команд. Регистр процессора, который указывает на программную инструкцию, которая будет выполняться следующей. Он отмечает смещение в кодовом сегменте.

Таблица векторов. Таблица указателей. Вектора прерывания. содержатся в 256 четырехбайтовых полях, занимающих младшие 1024 байта памяти. Каждое поле содержит адрес процедуры прерывания. INT 0 указывается первым вектором, INT 1 - вторым и т.д.

Таблица разделов. Таблица, содержащая главную запись загрузки на жестком диске. Она содержит информацию о размере и положении каждого раздела.

Таблица размещения файлов. Таблица, имеющаяся на каждом диске, которая хранит информацию о доступном дисковом пространстве и в которой записывается, какой кластер диска какому файлу отведен.

Тайм-аут. Выражение, используемое при операциях ввода/вывода, указывающее, что периферийное устройство не действует.

Текст программы. Исходный вариант программы, т.е. вид программы до того, как она была оттранслирована, ассемблирована или интерпретирована.

Текстовый файл ASCII. Последовательный текстовый файл, в котором все числа представлены в виде символов ASCII, а элементы данных разделены парой возврат каретки/перевод строки и конец файла отмечен символом ^Z (ASCII 26).

Текущий блок. Блок данных файла, состоящий из 128 записей, на который ссылаемся при доступе к файлу методом управляющего блока файла. *См. текущий номер записи.*

Текущий каталог. Каталог, являющийся частью дерева каталогов, к которому автоматически адресуются все файловые операции, до тех пор, пока строка пути в спецификации файла не указывает другого.

Текущий номер записи. При доступе к файлам методом управляющего блока файла данные организованы в блоки по 128 записей. Текущий номер записи - это номер записи в текущем блоке. Например, текущий номер для записи прямого доступа номер 128 будет равен 0, поскольку она будет первой записью в блоке 1 (весь отсчет начинается с нуля, поэтому запись с номером 128 будет 129-й записью файла, блок 1 - вторым блоком, а последняя запись блока 0 имеет номер 127).

Точка. Точка, выводимая в графическом режиме. В документации IBM ее называют также "pel".

Указатель. Переменная, которая содержит адрес другой переменной.

Указатель накопителя. Двухбайтовая строка, именующая дисковый накопитель, в виде A:, B: и т.д.

Управляющая строка. Строка символов, управляющая оборудованием. Управляющие строки часто включаются в поток данных, посылаемых на принтер или модем. Они начинаются со специального символа, указывающего их специальный статус (обычно символ ESC, ASCII 27).

Управляющий блок памяти. 16-байтовый блок параметров, создаваемый системой в начале каждого блока памяти, отведенного программе с помощью функций распределения памяти.

Управляющий блок файла. Блок параметров, создаваемый программой в памяти, для хранения информации, которая требуется системе для работы с файлом.

Управляющий код. Один из первых 32 символов набора кодов ASCII. Они обычно используются для управления оборудованием, а не кодирования данных. Наиболее часто употребляемыми управляющими кодами являются возврат каретки и перевод строки.

Устанавливаемые драйверы устройств. Драйвер устройства, который полностью интегрирован с системой, что позволяет ему использовать специальные средства проверки ошибок и управляющие средства.

Устройство. Вообще говоря, устройством называется любое оборудование, которое хранит, выводит или обрабатывает информацию, такое, как дисковый накопитель, видеодисплей или принтер.

Файловый указатель. Переменная, хранимая системой для каждого открытого файла. Файловый указатель отмечает позицию в файле, с которой будет выполняться следующая операция чтения или записи.

Физические координаты. Координаты точки на экране дисплея, отсчитываемые от левого верхнего угла, который имеет координаты 0,0. *См. также мировые координаты.*

Флаг переноса. Один из битов регистра флагов процессора, который часто используется функциями MS-DOS для индикации ошибки.

Флаги. Флаг - это переменная, которая может быть либо включена, либо выключена, сообщая о том, выполнено или нет определенное условие. Процессор имеет 16-битовый регистр флагов, в котором отдельные биты служат для индикации различных аспектов работы процессора.

Фоновые операции. Вторичный процесс, осуществляемый параллельно выполнению программы. Например, текстовый редактор может посылать данные на принтер, в то время как программа занята редактированием. Фоновые операции могут работать за счет использования прерываний.

Фоновый цвет. Фоновый цвет используется дисплеем. Это тот цвет, который принимает весь экран, когда он очищен.

Функция. В языках высокого уровня функцией обычно называют процедуру, которая преобразует данные из одной формы в другую. На уровне операционной системы словом "функция" называют любую из процедур обработки прерывания. Точнее, определенное прерывание может выполнять несколько процедур, каждая из которых называется функцией этого прерывания (номер функции всегда помещается в регистр AH при вызове прерывания). Сами функции могут содержать ряд подфункций.

Частота автоповтора. Скорость, с которой клавиша клавиатуры повторяет посылку кода, когда она держится нажатой.

Циклическая очередь. Тип буфера данных, в котором данные вставляются с одного конца, а берутся с другого. Текущие положения этих двух концов постоянно меняются и два указателя хранят текущие положения "головы" и "хвоста".

Циклический контроль четности. Метод проверки ошибок, в котором за переданным блоком данных следует вычисленный математически результат; после приема вычисление повторяется и сравнивается с переданным, чтобы быть уверенным, что данные переданы без искажения.

Цилиндр. У дисковых накопителей цилиндром называется группа дорожек, находящихся на одинаковом расстоянии от центра диска или дисков, помещенных в накопитель.

Эхо. Возврат для проверки. Например, при вводе с клавиатуры обычно выдается эхо на экран, выдается эхо и при выводе через коммуникационный канал.

Язык ассемблера. Язык программирования самого низкого уровня, в котором программист пишет инструкции, непосредственно управляющие работой процессора.

ОГЛАВЛЕНИЕ

Предисловие к русскому изданию 5
Введение 7

Глава 1. СИСТЕМНЫЕ РЕСУРСЫ 13
1.1. Ревизия системных ресурсов 13
1.2. Управление прерываниями 31
1.3. Управление программами 42

Глава 2. ТАЙМЕРЫ И ЗВУК 63
2.1. Установка и чтение таймера 63
2.2. Создание звука 85

Глава 3. КЛАВИАТУРА 110
3.1. Управление клавиатурой 110
3.2. Доступ к отдельным клавишам 137
3.3. Сводка кодов клавиш и их назначение 151

Глава 4. ВЫВОД НА ТЕРМИНАЛ 164
4.1. Управление выводом на терминал 164
4.2. Управление курсором 192
4.3. Вывод символов на экран 205
4.4. Вывод точечной графики 228
4.5. Сдвиг экрана и страницы 270

Глава 5. ДИСКОВЫЕ НАКОПИТЕЛИ 283
5.1. Управление распределением диска 283
5.2. Работа с каталогами диска 295
5.3. Подготовка к работе с файлами 312
5.4. Чтение и запись файла 336

Глава 6. ПРИНТЕР 384
6.1. Управление работой принтера 384

6.2. Установка спецификаций печати 395
6.3. Посылка данных на принтер 411

Глава 7. ВВОД/ВЫВОД 432
7.1. Доступ к последовательному порту 432
7.2. Создание драйвера устройства 460
7.3. Использование специальных устройств ввода/вывода 478

Приложение А. Двоичные и шестнадцатиричные числа и адресация памяти 491
Приложение Б. Битовые операции в Бейсике 496
Приложение В. Основные сведения об языке ассемблера 500
Приложение Г. Включение ассемблерных процедур в программы на Бейсике 507
Приложение Д. Использование драйвера устройства ANSI.SYS 509
Приложение Е. Набор инструкций для микропроцессора 8088 510
Приложение Ж. Набор инструкций для микропроцессора 80286 515
Глоссарий 519

Научное издание

Джордейн Роберт

Справочник программиста персональных компьютеров типа IBM PC, XT и AT

Книга одобрена на заседании секции редсовета по электронной обработке данных в экономике 24.02.89

Зав. редакцией А.В. Рассказов
Редакторы Т.Т. Гришкова, Е.В. Крестьянинова
Худ. редактор Ю.И. Артюхов
Ред. изд. системы Т.А. Козлова
Техн. редактор Г.А. Полякова
Корректоры Г.А. Башарина, Г.В. Хлопцева, Н.А. Поэгле
Переплет художника Е.К. Самойлова

Набор и репродуцируемый оригинал-макет изготовлен МП "Гемма"

ИБ № 2691

Подписано в печать 21.08.91. Формат 60 × 88 1/16. Бум. офсетная. Гарнитура "Таймс". Печать офсетная. Усл. п. л. 33,32. Усл. кр.-отт. 33,32. Уч.-изд. л. 31,81. Тираж 100 000 (2-й завод 50 001—100 000). Цена договорная. Зак. 295.

Издательство "Финансы и статистика", 101000, ул. Чернышевского, 7.

Тип. им. Котлякова издательства "Финансы и статистика" Министерства информации и печати СССР 195273, Санкт-Петербург, ул. Руставели, 13.

Р. Джордейн СПРАВОЧНИК программиста персональных компьютеров типа IBM PC, XT и AT

(43-50)

(43-50)